PDOS Realtime Operating System

Tips& TechnicalNotes

Vol. 1 No. 1
January 31, 1986

INTRODUCTION

The PDOS Tips and Technical Notes newsletter is intended to give you, the PDOS user, a new resource of valuable information. Since we are continually trying to improve on our product and the service to our customers, we will keep you up-to-date on the status of our product line. Other items featured here include warnings and cautions to avoid programming difficulties, fixes, patches, or work-arounds to known problems with current software, and special applications to make your programming job easier.

If you have items which would be of value to other users, we would like to hear from you. Please provide sample listings or disks containing the information you wish to convey.

Of course, the PDOS hotline remains a resource to help you in the solution of immediate problems through which we try to respond with answers to your difficulties as soon as possible.

CURRENT PRODUCT STATUS

Part #	Product Name	Current Version	Next Release
3510	PDOS 68000	3.0b	
3510-3/M	Force CPU-1,2,3 Installation Guide	11/8/85 Rev. B	
3510-4/M	VME-10 Installation Guide	11/15/85	
3510-4C/M	VMEsystem 1000 Installation Guide	1/31/86	
3510/M	PDOS Reference Manual	10/3/85 Rev. C	
3510/M1	Getting Started with PDOS	10/15/85 Rev. A	
3520	PDOS 68000 BASIC	3.0b	
3520/M	PDOS BASIC Reference Manual	10/1/85	
3530	PDOS 68000 PASCAL	2.7A	1st qtr. '86
3530/M	PDOS PASCAL Reference Manual	11/21/84	

Vol. 1 No. 1
January 31, 1986

INTRODUCTION

The PDOS Tips and Technical Notes newsletter is intended to give you, the PDOS user, a new resource of valuable information. Since we are continually trying to improve on our product and the service to our customers, we will keep you up-to-date on the status of our product line. Other items featured here include warnings and cautions to avoid programming difficulties, fixes, patches, or work-arounds to known problems with current software, and special applications to make your programming job easier.

If you have items which would be of value to other users, we would like to hear from you. Please provide sample listings or disks containing the information you wish to convey.

Of course, the PDOS hotline remains a resource to help you in the solution of immediate problems through which we try to respond with answers to your difficulties as soon as possible.

CURRENT PRODUCT STATUS

Part #	Product Name	Current Version	Next Release
3510	PDOS 68000	3.0b	
3510-3/M	Force CPU-1,2,3 Installation Guide	11/8/85 Rev. B	
3510-4/M	VME-10 Installation Guide	11/15/85	
3510-4C/M	VMEsystem 1000 Installation Guide	1/31/86	
3510/M	PDOS Reference Manual	10/3/85 Rev. C	
3510/M1	Getting Started with PDOS	10/15/85 Rev. A	
3520	PDOS 68000 BASIC	3.0b	
3520/M	PDOS BASIC Reference Manual	10/1/85	
3530	PDOS 68000 PASCAL	2.7A	1st qtr. '86
3530/M	PDOS PASCAL Reference Manual	11/21/84	

Part #	Product Name	Current Version	Next Release
3550	PDOS 68000 C	1.2c	
3550/M	PDOS C Reference Manual	10/1/85 Rev. A	
3560	PDOS 68K Absoft FORTRAN 77	2.1	
3560/M	FORTRAN Reference Manuals	12/1/85 Rev. A	
3511/M	Run Module Manual	10/1/85	
3410	PDOS 9900	2.4d	
3410/M	PDOS Reference Manual (9900)	1982 Rev. D	
3410/N	Update Notice		
3420	PDOS 9900 BASIC	2.4d	
3420/N	BASIC Installation Notes	2.4d	
3430	PDOS PASCAL	2.7A	1st qtr. '86
3430/M	PASCAL Reference Manual	1984	

WARNINGS AND CAUTIONS

1. If you are using the VMEbus, you should be aware that daisy chain jumpers must be installed, or all cards must be installed sequentially on the bus. Failure to do so may result in a system halt, or the device may not be located and available to the user.

2. Users who are developing 68K run modules should be aware that programs run slower in EPROM than in RAM. As a result, you may experience some timing differences from RAM tested and EPROM run programs.

3. CAUTION: Before upgrading to a new PDOS BASIC version, be sure to convert all BX files to the EX format and save a backup. The BX format may not be compatible with the new version.

4. With the 68K PDOS 3.0 release, it is necessary to use MASM R3.0b 10/17/85 and QLINK 11/12/85 versions. If you use 2.6 versions of these utilities, you will encounter problems.

5. The C 1.2c compiler produces self-relocating code, but not position independent code.

6. Currently in C Rev. 1.2c, only un-buffered I/O routines are implemented. All of the entry points for the buffered I/O routines (fopen, fclose, fputs, etc.) are set up and function as expected, except that the I/O goes to the disk immediately. "fflush" is not in the library -- it would be a no-op if it were. Buffered I/O will be implemented in a future revision of C. The most efficient I/O is through use of the routines in XLIB. XGLU reads an entire line from the console, letting the operator to perform command line editing before hitting return. XPLC dumps an entire string to the console. XRBF and XWBF read and write large blocks of data to the disk. XRLF reads a line from a disk file (delimited by a carriage return). XWLF writes a null-terminated string to the disk.

7. C external symbols must currently be unique in the first seven characters.

8. C initialization of multi-dimensioned arrays of structures is wrong.

9. There are a few other problems with combinations of structures with array fields with C. The compiler will sometimes generate bad code to address into such a data item.

10. When you want to open a device driver with the C FOPEN command, do not use "w" mode, since that will attempt to set the end of file mark to the beginning of the file (an illegal operation on a driver.) Instead, open it in "r" mode and write to it anyway.

11. The C Rev. 1.2c 'lseek' routine uses the XRFP--Read File Position primitive that is new in PDOS 3.0. If you use 'lseek' the code will not run on versions of PDOS earlier than 3.0. Also, 'fopen' with mode "a" or "a+" uses 'lseek' so the same warning applies.

12. If you try to assign a C constant 0x8000 to a long variable, the number will be sign-extended and 0xffff8000 will actually be assigned. This problem occurs because if a numeric literal will fit in sixteen bits, it will be stored as a sixteen bit constant and sign-extended on assignment. Leading zeroes do not help -- 0x08000 is the same as 0x8000. You must put a capital 'L' after the literal to force the compiler to create a 32-bit literal. Thus, assigning 0x8000L will give you the value you need. This problem, of course, extends to all numeric literals where the sixteenth bit is set. Thus, 32768-65535 or 0x8000 - 0xffff are affected.

13. Versions of C068 prior to 11/25/85 did not properly do a sizeof on literal strings.

14. The Rev. 1.2c C compiler requires about 85K to run. Currently it does not properly detect an attempt to run with too little memory. Versions of CSTART:ASM prior to 11/26/85 did not properly handle an out of memory problem. All C programs, therefore, suffered from the defect that they could be loaded into memory and then have the variable space run out of the task space. When this task space is cleared, it may wipe out the task's own stack, or worse, the TCB of the next task in memory. This also means that the compiler itself could crash the system if it were run in too small of a memory. Running the C compiler in too small of a memory (such as 32K) can crash the whole system, requiring a boot.

15. When creating C Rev. 1.2c EPROM programs, you should currently be aware of using functions that do dynamic memory allocation. In an EPROM program, it is not necessarily the case that the available memory lies between the __eomem and the bottom of the stack -- indeed, the stack pointer may be on the other side of the end of memory pointer. The situation can be fixed by dynamically loading the __eomem pointer with a value known to be down in the stack and assuring that the task is assigned sufficient stack space on start-up. In the meantime, the following routines (which all use dynamic memory allocation) should not be used in EPROM or should at least be very suspect: GLOB, COPY, FOPEN, TTYOPEN, XEQ, SYSTEM, SBRK, MORECOR, ALLOC, MALLOC, REALLOC, CALLOC.

16 There must be sufficient disk space available for the C compiler to create the intermediate files and output files. If this is not the case, the compiler may abort with an error 61 or some other peculiar error. In particular, the distribution disk does not have enough disk space to compile anything -- it is too full of code.

17. With FORTRAN Rev. 2.1a, a file error trapped with an "ERR=" on a read will show up again on the CLOSE if it does not also have an "ERR=".

18. If you attempt to produce both an assembly listing (/A switch) and a compiled source listing (/L switch) at the same time, you will only get the assembly listing since it includes the compiled source under FORTRAN Rev. 2.1a.

20. There are a few cases where invalid syntax will cause the FORTRAN compiler to crash. One of these cases is putting FORM='UNFORMATTED' in an INQUIRE statement.

21. Occasionally the FORTRAN run-time system will report errors and it is not immediately obvious whether the error is a PDOS error or a FORTRAN error. The program may report "COMMON buffer not found" when the error is actually "position error". Both are error #70 -- one from F77, the other from PDOS.

22. "USE option b" FORTRAN error message will come out even when you do use option b.

23. The FORTRAN rev 2.1a debugger will occasionally have trouble displaying the current value of a symbol, especially if you use the S(EARCH option to move into a different module and display common variables in that module.

24. If a FORTRAN subroutine calls another subroutine that was passed to it as a parameter (see EXTERNAL statement) the second subroutine is always loaded as an overlay, even if it has been linked in. Thus, the following three program segments execute just fine if allowed to link at run time, but will give an error 'Subroutine not found' if linked with F77L and DUMMY:SUB discarded.

```
------------ FIRST FILE ---------
       PROGRAM TEST
       EXTERNAL DUMMY
       CALL T1(DUMMY)
       END
------------ SECOND FILE ---------
       SUBROUTINE T1(SUBP)
       CALL SUBP
       RETURN
       END
------------ THIRD FILE ----------
       SUBROUTINE DUMMY
       WRITE(9,*) 'ENTERED DUMMY'
       RETURN
       END
```

25. The FORTRAN Rev. 2.1a compiler does not catch all syntax errors. One user found that the compiler did not flag a branch to a FORMAT statement label. Another error was when a variable name in a subroutine was declared as both a COMMON block variable and a passed parameter.

26. The FORTRAN Rev. 2.1a compiler generates position-independent code that runs at any address. However, there has been trouble loading programs into arbitrary address spaces and running them. This could mean problems when burning programs in ROM.

27. The Pascal compiler occasionally will report an 'OUT OF ADDRESS REGISTERS' error. Only 3 address and 5 data registers can be used by a program at a time. If the error occurs, compile the text with the O switch and find the area that must be rewritten. A typical program that will fail is as follows:

```
procedure m;
 type
    t = record  a : integer;  end;
 var
    c : integer;
     procedure e (f : integer ; h : t) ;
      function i : boolean ;
       begin{i}
        with h do {'with' statements may require an address reg.}
          i := f = c  {c is a global variable; f is a parameter,}
         end; {i}     {i is the function value; h is another }
     begin {e}        {parameter.  To handle all these different}
     end;  {e}        {addressing modes will cause the compiler }
 begin {m}            {to run out of addressing registers.}
 end;  {m}
```

FIXES, PATCHES, AND WORK-AROUNDS

1. Some 68K users have experienced difficulties when inputting messages longer than 64 bytes. You can fix this problem by changing the input buffer size in SYRAM to allow for 128 character messages. Use the following change in xxDOS:GEN and regenerate the system:

```
Change:    MASM MSYRAM:SR,#MSYRAM:OBJ;xxx
To:        MASM SYRAM:SR/IZ=7,#STRAN:OBJ;xxx
```

2. **>MTIME P,86** -- Some battery clocks do not keep track of the current year. So that your year will be correct, enter a second argument to the MTIME routine setting the PDOS year. Change your startup file to assure proper year upon startup of your system.

3. 68K MSYFL presently does not support the 'D' tag. You will get errors when trying to convert files using the new DCB.B data definition. These files can be converted by running QLINK, loading the file, and saving it.

```
QLINK
INPUT <FILENAME:OBJ>
OUTPUT <FILENAME>
SYFILE
END
QUIT
```

4. 68K PDOS BASIC programs with excessively long lines may give you problems when they are saved as BX files and later run. The long line halts loading or causes overwrite which could ruin the file. This is most likely to happen when transferring ASCII files from another system.

5. Some users have experienced difficulties following the use of the SPOOL command in 9900 BASIC. The SPOOL command needs to be reset and the SPOOL file closed. This can be accomplished with one of the following sequences:

   This resets the spool and closes all files:

```
SPOOL 0
RESET
```

   This resets the spool and closes only the spool file:

```
SPOOL 0
CLOSE MEMW[SYS(9) + 01E4H]
```

6. The following utility, MLIBGEN, was inadvertently not documented in the 3.0 PDOS Reference Manual. You might find it helpful to insert the page into your manual.

MLIBGEN
Library Generator Utility

Name: MLIBGEN

Function: Combines object files into a single library file

Format: **MLIBGEN**

Restrictions: MLIBGEN only builds new libraries. Existing libraries can be edited only by recreating them.

Description: MLIBGEN allows object files to be combined into a single library file. The entry (XDEF) labels for each library object are stored in the header of the library file along with the originating object file name and position of the library object within the library file.

When you specify a library load with the LIBRARY command during QLINK, PDOS will scan your files for any entry symbols that match any unresolved external (XREF) symbols in the link map. If a match occurs, then only the code corresponding to the XDEF label of the single library object is loaded. Thus, only those objects which resolve external symbols will be loaded.

Every time a library object is loaded, the LIBRARY command will start from the beginning of the library header and scan for new entries. It continues until no additional matches are found in the link map and library header.

Sample:

```
>MLIBGEN
68K LIBRARY GENERATOR 10/24/83
Copyright 1983, ERII
LIBRARY FILE=YOURLIB:LIB        The name of your library file
INPUT FILE=SUB1:OBJ             Origination object files to
INPUT FILE=SUB2:OBJ               become library objects
INPUT FILE=[CR]                 Type [CR] to end input files
ANY MORE FILES (Y/N)?N          Enter 'Y' to continue; 'N' to
                                  quit.
```

7. With C Rev. 1.2c, if you declare a global variable in two separately compiled modules but do not declare that variable in the main program, the linker will not know where to allocate space for that variable and will give an un-defined symbol message. A fix is anticipated for the linker, but until then, declare all global variables in the main module as well as in the other modules, or specify an initialization for variables defined only in subroutines. (Note: initialize it only in one module, or the variable will be doubly-defined!)

8. There are some bugs in the use of the "extern" keyword. In general, it is difficult in C Rev. 1.2c to distinguish between defining and referencing external variables. For now, this compiler takes all variables declared at the outermost level (with or without the "extern" keyword) to be definitions in the module where "main" is defined, and references if "main" is not defined. This does not hold true for static variables or for variables where an initialization is specified.

9. The C Rev. 1.2c 'printf' function may have problems printing integers larger than nine digits. It has an internal buffer of only 10 characters, and if a number (with the terminating null) exceeds this size, it overwrites other data. If this is a major problem, extract the 'printf' module from STDLIB:SRC, change the size of the buffer 'tbuf' to 12 characters, and use that new copy of the printf module by linking its object ahead of the STDLIB.

10. The following assignment creates bad code that causes a BUS ERROR under C Rev. 1.2c.

```
testl()
{
 float x[3],y[3];
 int i=0;
 asm("*y[i] = x[i] -- float");
 y[i] = x[i];
}
```

The workaround is to assign x[i] to a temporary and then assign the temporary to y[i]. The problem only shows up with floats; not with ints or longs, so you could cast the source and destination operands to long. This will be fixed in the next revision of C from Alcyon.

11. The XGML subroutine in C 1.2c has an error -- it doesn't report the proper value for the third parameter (last loaded address). The error can be fixed by extracting file XGML:ASM from the XLIB:SRC and making the following change:

    Change MOVE.L A3,D2 to MOVE.L A2,D2

12. It would be helpful if ROMLINK provided with C Rev. 1.2c allowed you to specify sections for RAM and ROM. The current program must be altered on line 96 (Sprintf statement puts out 'E' and '2' tags) and in the subroutine 'inrom' (returns 0 for ROM, 1 for RAM). If your section is greater than 9, you must also change the 'sprintf' statement where the '9' tag is output -- currently it goes out as a decimal digit and it should be hexadecimal. The following list shows the changes to make ROMLINK put the ROM code in section 14:

```
CHANGE:
sprintf(&line[27],"E0%08lxE1%08lx21%08lx2000000000",
TO:
sprintf(&line[27],"EE%08lxE1%08lx21%08lx2E00000000",

CHANGE:
sprintf(&line[linelen],"9%ld%08lx8",next->section,datal);
TO:
sprintf(&line[linelen],"9%lx%08lx8",next->section,datal);

CHANGE:
return(0);
TO:
return(14);
```

    Future versions of ROMLINK may accept command line parameters to set the sections to whatever is required.

13. The C 1.2c distribution version of LOCATE fails to create the bit map for programs larger than 64K. The problem is in the following statement:

```
    mapsize = ((unsigned int) mapptr >> 6) + 1;
```

This should be:

```
    mapsize = ((unsigned long) mapptr >> 6) + 1;
```

    This correction can be made on your system by changing file LOCATE:C and recompiling.

14. FORTRAN Rev 2.1a documentation for CRT:SA is lacking but can be found in the file.

15. The FORTRAN ENDFILE statement seems to be a no-op. Use the PDOS interface library functions to change the end of file mark.

16. The following code gives the FORTRAN Rev. 2.1a compiler problems. It reports a 'compiler synch error'.

```
CHARACTER *8 TEMP8
TYPE TEMP8(5:8),' '
```

Since this is an extension to F77, its use is questionable anyway, try:

```
TYPE (UNIT=9,FMT=*) TEMP8(5:8),' '
or
WRITE (UNIT=9,FMT=*) TEMP8(5:8),' '
instead.
```

17. Pascal Rev. 2.7a processes require more space on heap than is reasonable. This limits the total number of processes that can be created. Also, destruction of a process does not free up all of the space originally allocated. This is fixed with the new Rev. 2.8a.

18. Pascal 2.7a crashes when performing range checking (/R or $R switch) on a file where there is a case statement without an otherwise clause. This is fixed in Rev. 2.8a.

19. The following code causes an error in the assembler under Pascal 2.7a and has been fixed with 2.8a. The MASM error occurs because .ENDLOC is never XREFed in the file.

```
{$E}
PROCESS A;
BEGIN
END;

PROCEDURE B;
BEGIN
 A;
END;
```

20. The following code generates an error in the assembler text under Rev. 2.7a Pascal but has been fixed under Rev. 2.8a.

```
PROGRAM TEST;
VAR
 J : INTEGER;
 A : INTEGER;
 R : ARRAY [1..5] OF INTEGER;
BEGIN
        A := A + R[J];
        A := A - R[J];
END.
```

21. Pascal 2.7a attempts to rewrite to file 'TTA' which causes driver errors, or creates a new file called 'TTA' (not a driver) on the disk. This has been fixed with Pascal 2.8a.

22. Errors in specification of a 'WITH' statement argument such as WITH (A) DO cause the Pascal 2.7a compiler to crash. The parentheses are not valid and Pascal 2.8a will report this as an error.

23. Under Pascal 2.7a, using the EOF/EOLN functions cause the compiler to generate bad code. This is fixed in Rev. 2.8a.

24. Passing a string variable as a 'non-VAR' parameter to a routine that expected a larger string under Pascal 2.7a makes the compiler generate code to pad the string to the expected length. A bug in the compiler causes local variables in the calling routine to be corrupted after the call. This is fixed in Pascal 2.8a.

25. Various constructions involving large (greater than 32767 bytes) arrays causes the compiler to generate bad code under Pascal 2.7a. This is fixed under 2.8a.

APPLICATIONS AND HINTS

1. The following examples for converting decimal numbers to hex and hex to decimal illustrate the power of the PDOS operating system:

```
*DECIMAL TO HEX CONVERSION  (RESPONDS SIMILAR TO MONITOR COMMAND)
*
This example shows the interactive nature of the PDOS primi-
tives. Notice that there are no assembly code mnemonics in this
program except as assembler directives to establish the text
string for the output.
*
*       9900            68000
*       ======          =======
DH      XGNP            XGNP                    ;GET NEXT PARAMETER
        XCDB            XCDB                    ;CONVERT TO BINARY
        XCBH            XCBH                    ;CONVERT BINARY TO HEX
        XPMC            XPMC    EQ              ;PRINT ' = '
          DATA EQ
        XPLC            XPLC                    ;PRINT CONVERTED STRING
        XEXT            XEXT                    ;RETURN TO PDOS MONITOR
*
EQ      TEXT +' = '     DC.B    ' = ',0
        END DH          END     DH

*USAGE EXAMPLE:
>DH 256 = 0100          00000100
>DH 10  = 000A          0000000A

*HEX TO DECIMAL CONVERSION
*
*Uses only two assembly mnemonics and they are only used to
preclude the user from entering the hex descriptor in the input.
*
*       9900            68000
*       ======          =======
HD      XGNP            XGNP                      ;GET NEXT PARAMETER
        DEC R1          ADDA.L  #-1,A1            ;DECREMENT POINTER
        MOVB @HI,*R1    MOVE.B  #'$',(A1)         ;INSERT HEX DESCRIPTOR
        XCDB            XCDB                      ;CONVERT ASCII TO BIN
        XCBD            XCBD                      ;CONVERT RESULT TO DEC
        XPMC            XPMC    EQ                ;PRINT ' = '
          DATA EQ
        XPLC            XPLC                      ;PRINT CONVERTED STRING
        XEXT            XEXT                      ;EXIT TO PDOS MONITOR
*
EQ      TEXT +' = '     DC.B    ' = ',0
HI      BYTE '>',0
```

```
*USAGE EXAMPLE:
>HD 100 = 256           256
>HD A = 10              10
```

The previous examples can be entered and compiled to provide you with a helpful utility. Since these are programs, they will alter user memory. (PDOS monitor commands do not alter user memory).

2. Sometimes, it may be necessary to send special control characters to a printer from BASIC. The <null> and <tab> characters are not printable in PDOS BASIC since the <null> is a string terminator and the <tab> is replaced with spaces to the next print column. The following example provides a means for sending these and other character codes:

```
1    REM  CODE TO PRING SPECIAL CHARACTERS FROM BASIC
2    REM  NUMERIC VALUE OF CHARACTER IS PLACED IN COM(0)
3    REM  CHAR(0) CONTAINS ASSEMBLY CODE - USES XPDC TO PRINT CHAR
4    REM  CALL #ADR CHAR(0) PERFORMS PRINT

10   DIM CHAR[3]
20   $CHAR[0]=%'2E3C 0000 0001 224B D3FC 0000 0007 A096 4E75'
30   COM[0]=9              !SET FOR <tab>
40   CALL #ADR CHAR[0]     !PRINT IT

***
*68K ASSEMBLY WHICH GENERATES ABOVE STRING*
*
CHR     MOVE.L  #1,D7     ;SET PRINT FOR 1 CHAR
        MOVEA.L A3,A1     ;GET ADDRESS OF COM(0)
        ADDA.L  #7,A1     ;INCREMENT TO CHARACTER BYTE
        XPDC              ;PRINT IT TO CONSOLE
        RTS               ;RETURN
        END     CHR
```

Similar code in 9900 assembly can be created and entered on line 20.. Register R7 contains the address of COM[0].

3. The C compiler reports its errors by line number. Prior to May 1985, the PDOS editor did not easily allow you to find a particular line number. The new editor, of course, has a specific command to jump to a line number, but in the old editor (MJEDY), go to the top of the file ([CTRL-T]), set the jump count to one less than the desired line number ([ESC][CTRL-S]) and jump to that line from the top ([ESC][CTRL-J]). With the new editor, MEDIT, use the goto line function.

PDOS Realtime Operating System

Tips & Technical Notes

Vol. 1 No. 2
March 15, 1986

INTRODUCTION

We hope that last month's issue of PDOS Tips and Technical notes was of benefit to many of you, and we hope to continue this service on a regular basis. Any comments you may wish to make are appreciated.

NEW PRODUCTS

Several new programs are available for use under PDOS.

OWORD is a text runoff system for use with 9900 PDOS. Source code to OWORD is provided on the disk to assist the user in customizing the software to his hardware requirements. Source to OWORD is provided on the disk. This product is available "as is" and NO SUPPORT is provided.

Order Number: 3480-1 License Fee: $250.

STAT68 is an expanded statistical package available for use with PDOS on 68000-based systems with 700kb memory. It has graphics capability and handles the following statistical procedures and more: simple linear regression, polynomial regression, multiple regression, factorial analysis of variance, randomized fixed block analysis, Latin squares analysis, any factorial design with treatments being "crossed" and nested designs. Terminal support is available for HP-150, HP-2623. A preliminary manual is currently available.

Order Number: 3580-54 License Fee: $750.

WARNINGS AND CAUTIONS

1. The following errors and **oversights in our first newsletter** have been brought to our attention. Our apologies for any problems this may have caused:

   Warnings and Cautions:
   4. QLINK revision is dated 7/26/85. Also, hex QLINK entries require a '$' prefix.

   Fixes, Patches and Work-Arounds:
   1. was shown as : MASM SYRAM:SR/IZ=7,#STRAN:OBJ;xxx
      should be : MASM MSYRAM:SR/IZ=7,#MSYRAM:OBJ;xxx

2. **Assembly programs ending on odd boundaries** can cause errors at link time with QLINK. This will show up on the file following the file with the odd boundary. The odd boundary should be corrected with the EVEN directive of the MASM assembler.

3. Use caution with **disk buffering** on Stride, Mizar, VME/10, Hamilton Standard (formerly Mostek). When you write a file out (to a floppy as an example), while PDOS thinks it is written out, it probably IS NOT! The buffer has been altered, but the file has not been flushed to the disk yet. That means that you are at a risk to lose the data that you thought you saved. To be certain that you can safely remove a floppy, or turn off power, do a list (›LS) command for the Winchester before removing the floppy. You may see the floppy activity light come on, even if you are just reading the Winch. The best way to be sure that your information is indeed saved is to do a space (›SP) command on disks two and three. The following code could be included in a procedure file to flush the disk:

   ```
   SP 2
   SP 3
   RC
   ```

   On PDOS 3.0 you may use the xxPARK utility to flush the disk buffers before turning the computer off.

(WARNINGS AND CAUTIONS cont.)

4. If you plan on **creating tasks with high level language routines** in the PDOS operating environment, you should be aware that these higher level languages utilize high task memory to locate variables, or stacks. Should you create a new task and not have free memory available, you will be giving away some of your present task including variables, etc. If you intend to create tasks from higher level language tasks, be sure that you free sufficient memory prior to running the routines which will create the new tasks.

5. The 68000 RAM disk command allows you to specify the RAM disk to reside at any memory location giving you a high degree of flexiblity. However, PDOS makes no test to determine if the specified memory is already in use by PDOS or other tasks. Take care when **setting up a new RAM disk**.

   In one situation, the user specifed the new RAM disk larger than his free memory area. The PDOS mail array and Winchester disk buffers were changed. The result was a loss of directory information on the Winchester disk. On most 68000 systems, the RAM disk can be allocated at boot time by regenerating your system, specifying a larger RAM disk size, and re-installing the boot.

   For example, to generate a floppy sized RAM disk with 2560 sectors, you might use the following:

   ```
   0>xxDOS:GEN /RZ=2560
   ```

   The example above allocates the RAM disk from the TOP of memory. Be careful not to specify more 256-byte sectors than you have memory for.

6. When **using the >DM command** on 68000 PDOS, be sure to use the semi-colon and not the colon to delimit the level argument when the '@' symbol is used as a wild card. If a colon is used and the 'A' argument is used, all files on the disk may be deleted. If the ':' is used in the same manner in the >TF command, all files may be transferred.

   ```
            this colon should be a semi-colon
            v
   >DM @:FOR:@         DELETES ALL FILES!
   >TF @:FOR:@,0,A     TRANSFERS ALL FILES ON DISK 0!
   ```

(WARNINGS AND CAUTIONS cont.)

8. **ANSI terminal support** under rev 3.0b PDOS is an optional system parameter. If you wish to have ANSI terminal support, you must re-run xxDOS:GEN including the ANS option. You may wish to set switch CPSC to 0 so that ANSI will be your default terminal type. This means that MTERM will not have to be run. You will also need to reinstall your boot with MMKBT.

```
>xxDOS:GEN /ANS=1/CPSC=Ø,BASIC
```

Install ANSI terminal and set ANSI to default type. Include BASIC.

```
>MMKBT
 68K PDOS Make Boot Disk Utility Ø7/29/85
```

Select the (F)ile option to install the boot from your xxDOS file.

If you want to use Wyse-75 terminals with PDOS, set it in ATS mode and select the MTERM option letter "M", Data Media Excel 12 and not letter "D", Decscope (VT52).

C REV. 1.2C

9. C Rev. 1.2C -- **fscanf** currently cannot read across lines -- it seems to mess up when it hits a newline. For the moment, use fgets to put the data in a buffer and use sscanf to parse it.

10. C Rev. 1.2C -- **fprintf** has some sort of problem with %u when the F switch is set. The following program works (more or less -- the largest number should be 4 billion, not -2 billion) without the F switch, but not with it.

```
main()
{
    unsigned long x=1;
    int i;
    for (i=1;i<33;i++){
        printf("\n%lu,%ld",x,x);
        x += x;
    }
}
```

(WARNINGS AND CAUTIONS cont.)

11. When an **external reference appears next to a global definition**, a problem frequently occurs when someone includes "stdio.h" and then defines a few global definitions below it. As a temporary fix, rearrange the instructions (if possible) so that the two don't lie together. The bug shows up as an assembler error telling you that you 'XDEF'ed STDERR but you didn't define it.

12. Some of the **internal subroutines of the run-time library** (like .LMUL) are documented as if they were callable from a user program. In fact, although the symbols for the call are available (as .LMUL, for instance) those symbols are not made external via an XDEF statement. For the present, those subroutines cannot be called from a user program unless steps are taken to extract the sources from the library, insert the appropriate XDEF statement, and reassemble them.

PASCAL

13. 68000 Pascal Rev. 2.7A -- **closing a file** does not deallocate the file buffer F^. As a result, a series of OPEN/CLOSE statements will eventually run a system out of memory. This problem will be corrected in Rev 2.8B.

14. For all Pascal users -- **OPENing a file**, or any other operation involving NEW from within a PROCESS (rather than from a PROCEDURE/FUNCTION) will cause an out of memory error, number 603. This is because when the process is created, its stack is allocated from the heap. Later, when the code in the process requests memory from the heap, the runtime system notes that the current stack pointer is lower than the current heap pointer -- an indication that the heap and stack have overrun each other. For now, consider it a restriction that processes cannot perform operations involving dynamic memory allocation. This restriction will be lifted in the future.

FIXES, PATCHES AND WORK-AROUNDS

1. A correction should be noted for the **PDOS debugger documentation** on page 3-29 of the 3.0 PDOS Reference Manual. The explanation of the use of the trace "T" command indicates that "a return will execute it and display the next instruction to be executed." The "return" should be a "space."

2. On Stride systems, the **S6LDGO** program as supplied on the boot disk causes the system to hang. To correct this difficulty, assemble the S6LDGO:SR into S6LDGO using MASM and MSYFL.

3. When attempting to **download S-records to target systems**, some users have had some difficulty. Motorola's description of S-records indicate that each record may be terminated with a CR/LF/NUL. Force systems seem to make this a requirement. The following patch to our MSREC:SR utility will provide these line delimiters:

```
*
SREC12  ADDQ.W  #4,A7           ;Y, POP OVER TERMINATER
        MOVE.B  #$ØD,(A2)+      ; WAS $ØA
        MOVE.B  #$ØA,(A2)+      ; WAS $ØD
        MOVE.B  #$8Ø,(A2)+      ; ADDED NULL WITH HIGH BIT SET
        CLR.B   (A2)
        LEA.L   LBUF(PC),A2     ;POINT AT S-RECORD
        MOVE.L  (A7)+,D5        ;RESTORE COUNT
        RTS
```

   Use MASM and MSYFL to rebuild your new MSREC syfile.

   It is also important that the Force monitor RO register contain the offset to the desired load address; otherwise, the code may not be loaded.

4. The following utility, xxLDGO, is completely documented in the Installation and Systems Management guide for your system; however, the general information sheet on the next page may be useful for you if you do not have access to the guide. It has been prepared so that you might insert it into your PDOS Reference Manual.

xxLDGO
LOAD AND/OR GO TO A NEW SYSTEM

NOTE: xx should be replaced with the letters for your specific system (S6 for Stride 460, F1 for Force CPU-1, V2 for VMEsystem 1000, etc.)

Name: xxLDGO
Function: Load into memory and/or execute new system.
Format: ›xxLDGO {‹load address›}{,‹filename›}

Restrictions:
xxLDGO will replace your current PDOS operating system and execute a new system terminating all tasks.

Description:
xxLDGO is used to load and execute new PDOS systems.

The load address is the location in memory where the program is to be located.

The filename is the name of your system file. If a filename is not given, then xxLDGO will look for a PDOS system in your task space. xxLDGO will only load a file in which the PDOS ID characters are found. After xxLDGO has loaded your new system, it will jump to the load address and begin execution.

The following is an example:

```
     >xxLDGO ,xxDOS[CR]
```

You then see something similar to the following:

```
                DOS File Loaded: xxDOS
          Found PDOS at address $ØØØØBEB6
                    DOS size is $ØØØØ8BD4
```

Execution of the startup file on the new version. . .

```
     >MTIME P[CR]
        . . . .
```

xxLDGO allows you to try a new version of PDOS without modifying your disk boot image. To make this new system into a disk boot you need run the MMKBT utility.

5. A patch to include **HP-150 terminal support** is shown below. It may be included in MBIOS:SR. You may have either ANSI terminal support or HP-150 terminal support, but not BOTH. After making this patch you will need to re-run xxDOS:GEN and MMKBT to install your new system.

```
*
*       MBIOS SUBROUTINE FLAGS
*
        IFUDF   HP15Ø:  HP15Ø   EQU 1   ;DEFAULT INCLUDE HP15Ø
*
*
        IFNE    HP15Ø
*************************************************
*       HP 15Ø POSITION CURSOR
*
*       IN:     D1.B = Y POSITION (ROW)
*               D2.B = X POSITION (COL)
*               (A3) = CBØ$(A6)
*       OUT:      SR = .NE.
*
*       MODE = <esc>&aYYyXXC
*
B$PSC   MOVE.W  #$9B*256+$8Ø+'&',(A3)+
        MOVE.B  #$8Ø+'a',(A3)+
        CLR.L   DØ
        MOVE.B  D1,DØ           ;GET Y OR Row POSITION
        DIVU.W  #1Ø,DØ          ;HIGH = REMAINDER, LOW = QUOTIENT
        ADDI.B  #$8Ø+'Ø',DØ
        MOVE.B  DØ,(A3)+
        SWAP    DØ
        ADDI.B  #$8Ø+'Ø',DØ
        MOVE.B  DØ,(A3)+
        MOVE.B  #$8Ø+'y',(A3)+
        CLR.L   DØ
        MOVE.B  D2,DØ           ;GET X OR Col POSITION
        DIVU.W  #1Ø,DØ          ;HIGH = REMAINDER, LOW = QUOTIENT
        ADDI.B  #$8Ø+'Ø',DØ
        MOVE.B  DØ,(A3)+
        SWAP    DØ
        ADDI.B  #$8Ø+'Ø',DØ
        MOVE.B  DØ,(A3)+
        MOVE.B  #$8Ø+'C',(A3)+
        CLR.B   (A3)+
        CLR.W   -(A7)           ;SET A .NE.
        RTR
*
```

(FIXES, PATCHES, AND WORK-AROUNDS cont.)

```
*************************************************
*       HP-15Ø  - CLEAR SCREEN
*
*       HP-15Ø MODE = <esc>&aØyØC<esc>J
*
*       IN:
*       OUT:        SR = .NE.
*
B$CLS   LEA.L   HPCLR(PC),A2     ;POINT TO CLEAR SCREEN SEQUENCE
*
@ØØØ2   MOVE.B  (A2)+,(A3)+      ;OUTPUT, DONE?
          BNE.S @ØØØ2            ;N
*
        CLR.W   -(A7)            ;SET A .NE.
        RTR
*
HPCLR   DC.B    $9B,$8Ø+'&',$8Ø+'a',$8Ø+'Ø',$8Ø+'y'
        DC.B    $8Ø+'Ø',$8Ø+'C',$9B,$8Ø+'J',Ø
        EVEN
        ENDC

*
*
        IFNE    HP15Ø
          PRINT '>> HP 15Ø TERMINAL SUPPORT INCLUDED'
        ENDC
```

6. A patch to assure that **the battery clock year** matches the PDOS year is as follows:

Under label:	TIMR
Change:	LEA.L YEAR(PC),A1
To:	ADDA.L #6,A1

and reassemble the source

This patch allows the PDOS year to be used for the battery clock year rather than using the year of the last compile for the MTIME routine.

6. **Very large programs in 68000 BASIC** Rev 3.0b will have problems if more that 253 variables are used. When a line defining the 253rd variable is entered with BASIC's line editor, or when it is brought in from a file with the LOAD command, it is garbled. As an example, see the file below:

(FIXES, PATCHES, AND WORK-AROUNDS cont.)

```
2ØØØ A1=Ø: A2=Ø: A3=Ø: A4=Ø: A5=Ø: A6=Ø: A7=Ø:
2Ø1Ø A8=Ø:A9=Ø:A1Ø=Ø:A11=Ø:A12=Ø
. . .
247Ø A247=Ø: A248=Ø: A249=Ø: A25Ø=Ø: A251=Ø: A252=Ø
25ØØ REM
251Ø REM NOTICE THE NEXT LINE
252Ø REM
253Ø A253=Ø
```

When this file is LOADed, the assignment on line 2530 is garbled.

The fix for this problem is a patch in the 68000 BASIC interpreter itself. The following BASIC program searches for the appropriate location in your system and applies the patch.

```
1Ø   REM BASIC PATCH FOR 253rd VARIABLE PROBLEM
2Ø   REM
3Ø   REM CHANGE: E18C 72Ø3 E19C 4AØ4 67Ø2 18C4 5341 6EF4 4E75
4Ø   REM     TO: 4844 61Ø4 61Ø2 6ØØ4 4AØ4 67Ø2 18C4 E19C 4E75
5Ø   REM
6Ø   I=MEML[SYS[39]]  ! SELECT START OF BIOS FOR SEARCH
1ØØ   FOR A=Ø2ØØØH+I TO ØCØØØH+I
11Ø   IF MEML[A]=ØE18C72Ø3H: IF MEML[A+4]=ØE19C4AØ4H: GOSUB 1ØØ
12Ø   NEXT A
13Ø   PRINT "*** PATCH NOT FOUND!!! ***"
14Ø   BYE
2ØØ   IF MEML[A+8]=Ø67Ø218C4H: IF MEML[A+12]=Ø53416EF4H: SKIP 1
21Ø   RETURN
22Ø   MEML[A]=Ø484461Ø4H: MEML[A+4]=Ø61Ø26ØØ4H
23Ø   MEML[A+8]=Ø4AØ467Ø2H: MEML[A+12]=Ø18C4E19CH
24Ø   PRINT "*** PATCH COMPLETE!!! ***"
25Ø   BYE
```

After this patch has been applied, you may want to test it out by LOADing a program like the one above. If everything works okay, make the patch permanent by running MMKBT and selecting the "M" option for the source of the boot. See your installation manual for details on the operation of MMKBT on your particular system.

We recommend that you try out the boot by writing it on a temporary floppy disk first. If that works, you can install the boot on your hard disk by using similar procedures.

(FIXES, PATCHES, AND WORK-AROUNDS cont.)

7. All PDOS 3.0b systems with 68010 processors will experience a format exception error when the XKTB primitive is used on tasks that have been created with XCTB upper/lower memory bounds format. To fix this problem, take the address of the exception, add six, and change the contents.

```
   >
 FRMT exception with XKTB
   FRMT @ØØØØ2C64    < address
   DØ: ØØØØØ . . . .
   AØ: . . . .

 Enter debugger and change contents of exeception address + 6

   >PB
   2C64 4E73
   2C66 4FEE
   2C68 Ø3AE
   2C6A 67ØC 51D7     < new value
          ^
         old value
```

Run MMKBT with the (M)emory option to save the new boot.

This change will prevent the task abort feature (file MABORT) which has been implemented on some systems from working.

APPLICATIONS AND HINTS

1. Some PDOS users have terminals which allow up to 132 characters on the screen and/or more than 24 lines per screen. MEDIT Rev 1.9 or later allows you to **select row and column size**. The default is 80 columns and 24 lines per screen. To utilize this feature, you can use the two optional row and column arguments:

   ```
   MEDIT <filename>{<,col><,row>}

   >MEDIT ,132                for 132 columns
   >MEDIT FILENAME:SR,,49     for 8Ø columns and 49 lines
   ```

2. To create a task on a terminal **without displaying the PDOS prompt**, you can use one of the following procedures:

   a. Create the task on port zero and then reassign the port for the terminal within the operating task.

   or

   b. Create a dummy task on port zero, reassign the port for the terminal within this task, and chain to the desired task.

3. It is often desirable to **access certain PDOS variable buffers** such as the task control block or SYRAM from higher language routines. In PDOS BASIC, the SYS 9 function returns the address of the user task control block. SYS 39 returns the address of SYRAM.

   In FORTRAN, Pascal, or C, addreses to these buffers can be obtained by using the XGML primitive. You should refer the specific language manual for the use on this primitive.

4. To **pass long integer values to the Pascal XPSF routine** on 9900 systems as it is currently defined is not possible. As defined, one can access up to 32 Kbytes. Several approaches can be used to access further into the file. A routine which reads the record number and the bytes per record can be set up to index into files on an even number of records. The long integer is set up via a multiplication within the routine.

   An alternate procedure which requires combined Pascal and assembly procedures and will position to any byte within the record is as follows:

```
{PASCAL PROCEDURE WHICH CALLS SPECIAL POSITION ROUTINE}

Procedure XPSF1(FILID,MUL,ADD: INTEGER); EXTERNAL;

Procedure XPSFC(FILID: INTEGER; BYTES: REAL);
  VAR
    MUL, ADD : INTEGER;
  BEGIN
    MUL := TRUNC(BYTES/30000.0);
    ADD := TRUNC(BYTES-(MUL*30000.0));
    XPSF1(FILID,MUL,ADD);
  END;


*       TXPSF1:SR       13-FEB-86
****************************************************************
* PDOS SUPPORT ROUTINES FOR PASCAL TI9900  (future)
* (C) 1984 ERII, PROVO UT
****************************************************************
* ROUTINE NAME: TXPSF1
* FUNCTION:     Positions a file to a specified byte index
* REV:          2.8a
* AUTHOR:       David A. Grotegut
*
*  ASSEMBLY PROCEDURE TO POSITION TO BYTE IN FILE
*  USES MULTIPLIER * 30000 + ADDER PASSED BY PASCAL ROUTINE
*
        COPY TPHEAD:SR
*
        PSEG
        IDT '2.7TXPSF1'
        DEF XPSF1
        REF .PERROR
*
*PROCEDURE  XPSF1(FILID, MUL, ADD: INTEGER); EXTERNAL;
*
SP      EQU R10
*
TXPSF1  DECT SP
        MOV *SP,R0              ;GET ADDER
        DECT SP
        MOV *SP,R2              ;GET MULTIPLIER
        DECT SP
        MOVE *SP,R1             ;GET FILID
        MUL  @CONST,R2          ;EXTEND TO LONG INTEGER
        ADD  R0,R3              ;ADD EXTRA BYTES
        XPSF                    ;POSITION TO BYTE
         JMP TXERR
```

(APPLICATIONS AND HINTS cont.)

```
TXERT   RT
*
TXERR   MOV R11,*SP+            ;SAVE RETURN
        AI RØ,ERHIGN+ERHLOC     ;IGNORE THE ERROR BUT REPORT IT
        BL @.PERROR
        DECT SP
        MOV *SP,R11             ;RESTORE RETURN
        JMP TXERT               ;RETURN ANYWAY
*
CONST   DATA 3ØØØØ
        END
```

5. **PDOS BASIC will interpret hex strings** and output the proper character string when saved as a string variable i.e. $A= '‹1B›*'. String variables are still string variables and cannot be compared with another string although they may produce the same output. For example: $A = '‹41›', $B = 'A'. If $A and $B are printed, they will produce the character A but they are not the same string. Use $A = %65%0 to be equivalent with $A = 'A'.

6. It is possible to **have a 9900 BASIC program running and be able to execute a monitor command from a keyboard** as though the BASIC program were not there. The following two lines of code will perform the application:

```
1ØØ  BASE SYS 16: CRB 18=Ø: $INTR=%'Ø42Ø ØØ1Ø Ø45B': MEMW Ø342H=Ø1F15H
11Ø  IF CRB 21: CALL #ADR INTR: MEMW Ø342H=Ø1F1ØH: BYE
```

By way of explanation, BASE SYS 16 gets your console CRU base. CRB 18=0 disables interrupts on receive for that port. The assembly language string is a BLWP to the interrupt service routine followed by a RT return. The MEMW modifies the interrupt service routine to check RBRL instead of RINT.

Note: Enabling this feature can have some side effects for other ports in the system which have their interrupts disabled but are still receiving characters. This is the case especially if they are higher in the task list than your console.

The second line needs to be executed often in the application. It will normally fall through unless a character is received. A character in the receive buffer of the 9902 will cause the modified interrupt service routine to put the character in the input buffer. Upon return from the interrupt service routine, the modification is removed and BASIC exits. The monitor then gets the received character and any that follow. The address of the (1F10) instruction may be different on your system but it should be near to the 0342H address.

7. **68000 SECTION labels in MASM and QLINK** are used to group sections of code together. Files of code containing section lables will be grouped together as they are noted by the assembler or linker even if they are from a separate include file. For example:

```
SECTION Ø                  Will compile to          SECTION Ø
                           the following
CODE A                     sequence:                CODE A
CODE B                                              CODE B
                                                    CODE C
SECTION 2                                           CODE E

CODE D                                              SECTION 2
CODE F
                                                    CODE D
SECTION Ø                                           CODE F

CODE C                                              SECTION 3
CODE E
                                                    CODE G

SECTION 3

CODE G
```

8. There have been questions about using the **error trapping feature in Pascal** to catch various types of run-time errors.

   The following program illustrates trapping the PDOS error 53, "File Not Found," to check whether or not a particular file exists. This program could be used to validate user input, search a directory for files, or to determine whether to create a new file or append to an existing one. Similar techniques could be used to trap the other Pascal run-time errors.

```
PROGRAM TEST;
CONST
  ERHIGN=Ø;           {"IGNORE" signal}
  ERHABT=2;           {"ABORT" signal}

VAR
  EXISTS : BOOLEAN;
  FILENAME : STRING[24];
  MYFILE : TEXT;

PROCEDURE SETERR(PROCEDURE EH(VAR E,A:INTEGER));EXTERNAL;

PROCEDURE ERTRAP(VAR E,A:INTEGER);
BEGIN {ERTRAP}
  IF E=53 THEN BEGIN
      EXISTS := FALSE;        {File does not exist}
      A := ERHIGN;
    END
  ELSE
    A := ERHABT;
  SETERR(ERTRAP);             {Restore error trap}
END;  {ERTRAP}

BEGIN
  SETERR(ERTRAP);
  REPEAT
    WRITE('ENTER FILE NAME ');
    READLN(FILENAME);
    EXISTS := TRUE;           {Assume that it exists}
    RESET(MYFILE,FILENAME);
    IF NOT EXISTS THEN WRITELN('NOT THERE');
  UNTIL EXISTS;
  CLOSE(MYFILE);
END.
```

9. **9900 Pascal users** who are writing or calling assembly code routines which reference variables in the status block should include the following code in their routines:

```
REF      .PTCB           ;EXTERNAL REFERENCE
.
.
.
MOV @.PTCB(15),R9        ;GET STATUS BLOCK ADDRESS
```

68000 Pascal users can also use the following to obtain their task control block:

```
XREF     .PTCB             ;EXTERNAL REFERENCE
.
.
MOVEA.L .PTCB(A4),A6      ;GET STATUS BLOCK ADDRESS
```

If you fail to do this and the program uses R9 or A6 to reference a status control block variable, you could be referencing an undefined location which may cause other tasks or your system to crash. Follow the guidelines for register usage in section 5 of the PDOS Pascal Reference Manual.

10. **Accessing System Memory as Fortran Variable Space.**

On occasion, it is necessary to write Fortran programs that share some sort of data space. What you need is some sort of COMMON that extends across task boundaries, or some way of sharing memory between tasks. The FORTRAN 77 language specification does not offer any way of doing this; indeed, it does not even allow for the concept of a "task," but with PDOS Fortran and a little imagination, it can be done.

The trick is to take advantage of the Fortran feature that passes all parameters by address. If you pass an array into a subroutine, that subroutine really receives just a pointer to the beginning of the array and makes all references to the array indirectly through that pointer. What if that pointer really pointed to the global variable space? Then accesses to the elements of the array would really be reading and writing that memory out somewhere in system RAM!

You can get the address of a block of memory through various techniques. Perhaps the easiest way is to free memory with the PDOS >FM command and note the address that it prints out. This is the address of the system memory you will use. You could write that address explicitly into your programs, or have them read it from some sort of file. Or, you could put the address away where everyone can easily get it -- such as in the MAIL array.

The following program illustrates the use of the XGML call to get the address of the MAIL array. Then, rather than allocate memory from the system, dedicate a long word of the mail array itself as the variable space and use another word of the mail array to point to that space. That space is set to a value of 100 in this example, so that it can be examined later to see if you got it properly.

(APPLICATIONS AND HINTS cont.)

Since the mail array is only 256 bytes long, you wouldn't use it for large collections of data, but allocate them elsewhere and just leave a pointer here.

It is normally safer to skip the very beginning of the mail array (this example uses the starting address plus 8) because BASIC tends to use that location for its own purposes.

```
PROGRAM TEST2
IMPLICIT INTEGER (A-Z)
EXTERNAL A,PASSER
CALL XGML(ENDTCB,UPPERMEM,LASTLOAD,SYRAM,TCB)
MAIL = LONG(SYRAM+4)
LONG(MAIL+8) = MAIL+16   ! MAIL(Ø) OF BASIC MAIL ARRAY
LONG(MAIL+16) = 1ØØ      ! MAIL(1) OF BASIC MAIL ARRAY
END
```

Now that you have set up the mail array, you need to call a subroutine and pass the address of the global data to it. You will have to use an assembly language routine. And since the assembly language routine will need to know the subroutine to call as well as the address to pass it, you should pass both to it. That way, you can call Fortran routines from assembly language.

```
PROGRAM TEST1
IMPLICIT INTEGER (A-Z)
EXTERNAL A,PASSER
CALL XGML(ENDTCB,UPPERMEM,LASTLOAD,SYRAM,TCB)    !get address of SYRAM
MAIL = LONG(SYRAM+4)                             !get address of MAIL
GLOBAL = LONG(MAIL+8)                    !get address of global data

C
C       Send address of global data to subroutine A via PASSER
C
C

CALL PASSER(GLOBAL,A)
END

SUBROUTINE A(I)
WRITE(9,*) 'I = ',I      ! display value of global data
RETURN
END
```

(APPLICATIONS AND HINTS cont.)

This assembly language routine receives the pointer to the global memory and the address of the subroutine to call. It then calls the one with the address of the other.

```
PASSER  MOVEA.L 4(SP),A2           ;GET  ADDRESS  OF  ADDRESS  OF  SUB-
                                   ROUTINE 'A'
        MOVEA.L (A2),A2            ;GET ADDRESS OF SUBROUTINE A
        MOVEA.L 8(SP),A1           ;GET ADDR OF GLOBAL VAR
        MOVE.L  (A1),-(SP)         ;GET VALUE  OF GLOBAL  VAR (WHICH IS
                                   AN ADDRESS)
        MOVEQ.L #1,DØ              ;ONE ARGUMENT TO BE PASSED
        JSR     (A2)               ;CALL THE SUBROUTINE
        ADDQ.L  #4,SP              ;CLEAN UP STACK
        RTS                        ;AND RETURN
        END
```

The remaining difficulty is to make everything work together. This involves the use of the Fortran compiler, the Fortran linker, and (to prepare the PASSER routine) the PDOS assembler and SY file converter. The programs TEST1:FOR and TEST2:FOR are compiled in a normal fashion. The command lines might be:

```
x>F77 TEST1
x>F77 TEST2
```

The file PASSER:SR must be prepared by assembling it and converting it to an SY file like this:

```
x>MASM PASSER:SR,#PASSER:OBJ
x>MSYFL PASSER:OBJ,#PASSER:SUB
```

The Fortran programs must now be linked with the different support routines like this:

```
x>F77L TEST1,PASSER,XLIB/L,F77:RL/L
x>F77L TEST2,XLIB/L,F77:RL/L
```

You must run TEST2 first to set up the pointer in the MAIL array.

```
x>TEST2
```

Now, when you run TEST1, it will print the value that was stored by TEST2.

```
x>TEST1
I = 100
```

Good luck!

PDOS Realtime Operating System

Tips& TechnicalNotes

Vol. 1 No. 3
May 5, 1986

INTRODUCTION

New Release

Fortran 2.2 for the 68000 is now available. There were some changes since the 2.2 beta release. The release consists of a new disk and release notice. The current manuals are unchanged. If you desire an upgrade, please contact Karen Vanfleet at Eyring.

Included in this issue are the following items:

WARNINGS AND CAUTIONS

1. **CAUTION -- 68000 PDOS 3.0b XSUI.** The primitive when used to suspend a task on event reset may not work as you expect.

   The XSUI primitive suspends a task until either one or two events occur. In order to suspend on one event, the upper byte of D1.W must be set to zero with the lower byte containing the desired event. The event number bytes are positive if you want to suspend until the event is SET (-1). The byte is negative to suspend until the event is RESET (-0). In asembly, when you MOVE.W, MOVE.L, or MOVEQ.L a negative number to a register, the low byte contains the negative byte, and the other bytes are set to $FF.

   For example, MOVE.W #-32,D1 yields D1 - $FFE0 and MOVE.L or MOVEQ.L #-32,D1 yields D1 - $FFFFFFE0. If these instructions are used wtih the XSUI call, there is an $FF byte for the second event telling PDOS to suspend until either event 32 is RESET (-0) or until event 1 is RESET (-0). Since event 1 defaults to RESET, then the task calling XSUI never suspends, regardless of event 32. To solve the problem in assembly, just mask off the upper byte with MOVE.W #-32&$00FF,D1.

   The problem is more subtle in Pascal. For example, the statement

   ```
   XSUI(Temp,-32)
   ```

   only suspends until either event 1 or 32 are RESET. Since event 1 comes up RESET, the task never suspends. You can work around this problem by setting event 1 when booting your system (i.e. >EV 1) in the SY$STRT file.

   To work around the problem in Pascal, always place a single negative event number into the higher byte as follows:

   ```
   XSUI(Temp,-32*256);
   ```

2. **CAUTION -- Using MFSAVE.** If you attempt to utilize the MFSAVE utility to recover a file which you have inadvertently deleted, be sure to save it to an already existing file on the same disk or create a new file on a separate disk. If the file is created on the existing disk, it may utilize the first sector of the file being saved. As a result, at least one sector would be destroyed as the new file is created.

FIXES, PATCHES AND WORKAROUNDS

1. **FIX -- Pascal Procedure Files.** Some versions of 68000 PASCAL Rev. 2.6c may encounter an error when running the procedure file "PASCAL". The following command line:

```
INPUT PTEMP&#W:POB
```

should be changed as follows:

```
INPUT PTEMP&#:POB
```

2. **FIX -- FxBIOSU Parity Enable.** An error in the FxBIOSU files for the FORCE CPU-1, CPU-2, and CPU-3 prevents the system from using the SIO card with EVEN parity. The port "locks up." Even though the UART is correctly initialized for parity, the input interrupt is disabled and never enabled.

The following code in FxBIOSU:SR is the culprit:

```
BTST    #5,D1              ;ENABLE?
  BNE.S @006               ;N
TAS.B   RIER(A0)           ;Y, ENABLE INTS
```

Change the '5' in the first line to 'BRIN' so that it reads:

```
BTST    #BRIN,D1           ;ENABLE?
  BNE.S @006               ;N
TAS.B   RIER(A0)           ;Y, ENABLE INTS
```

Then, "GEN" the system with FxDOS:GEN and check it again. You should now be able to communicate through the port with or without even parity.

3. **FIX -- DISK ACCESS ON VME 120.** The selected drive may not be accessed when more than one floppy drive is installed in VME 120 applications. To correct this problem, the following fix should be implemented:

Under the label W$XDIT in V2BIOSW:SR

```
change    CMPA.L  D0,A2          to      CMPA.L  D0,A2
            BEQ.S @020                     BEQ.S @030
```

Regenerate your system with this correction.

4. **PATCH -- XDEV Under 3.0b and Later.** XDEV on 68000 PDOS 3.0b or later may not set events when the system clock interrupts and rolls the event delay queue.

   The following patch should correct this problem:

   A. Reboot and kill all tasks except task 0.

   B. Using the debugger, alter the following to disable interrupts during the XDEV call:

Old Hex	Old Assembly	New Hex	New Assembly
4AED00BE	TAS.B TLCK.(A5)	007C0700	ORI.W #$0700,SR
5BC7	SMI.B D7	4207	CLR.B D7
4401	NEG.B D1	4401	(NO CHANGE)

   To make the alteration, enter the PDOS debugger and find the address of the long word $5BC74401:

```
>PB
800,9000,5BC74401L
  001FD4
```

   Only one address should be listed. If there are more than one, use the first one. With that address, open the location with a carriage return. Use the minus sign to step backwards two locations and enter the new code:

```
1FD4[CR]: 5BC7-
1FD2 00BE-
1FD0 4AED 007C[LF]
1FD2 00BE 0700[LF]
1FD4 5BC7 4207[CR]
Q                             return to PDOS
>
```

   Interrupts are enabled when the XDEV primitive returns to your task.

   C. Once the patch is made, you should save the patch using MMKBT with the M(emory) option.

   This problem will be fixed in a future release and we apologize for any difficulty it may have caused.

5. **PATCH -- MEDIT For Lines Longer Than 96.** As reported in the previous issue of PDOS Tips and Technical Notes, a hidden problem in MEDIT prevents the use of lines longer than 80 characters.

   To patch the editor so that it will handle longer lines, do the following:

```
>ZM
>LO MEDIT                                         Load MEDIT program
>LT                                               Note TB and BM addresses
Task   Prt Tm  Event   Map Size   PC    SR    TB     BM     EM    I 1 2 4 8
*0/0   64  1            0  638  001E94 2000 00C000 00EBC0 0AB800  1 1 4 0 0
 1/0   64  1   98       0  200  002686 2004 0BC000 0BEBC0 0EE000  2 2 4 0 0
>PB                                               Enter debugger
+0,FFFF,0C420050L                                 Search for $0050
  00D48                                           Note address
DD48[CR]0C42  B46E[LF]                            Enter address from above and enter
                                                [CR]
DD4A    0050  1DDE[CR]                            Change  next  address  to  one
                                                calculated above

Dump and disassemble to verify instruction change to CMP.W $1DDE,D2

Q                                                 Quit the debugger
>SV #T,$C500,$EBC0[CR]                            Save modified MEDIT
>T FILE,132                                       Try modified version

(***THE CODES IN THE ABOVE EXAMPLE MAY VARY FROM SYSTEM TO SYSTEM***)
```

   Before the modified editor can be used you should set up the system to handle a modified ANSI terminal to output the proper cursor control sequence. The following example will help you with the change:

```
**************************************************
*        (WY-50) - POSITION CURSOR
*
*       IN:     D1.B = ROW POSITION
*               D2.B = COLUMN POSITION
*               (A3) = CB0$(A6)
*       OUT:      SR = .NE.
*
*        MODE = <esc>arrRcccC
*
        IFNE    ANS
B$PSC   MOVE.W  #$9B00+$80+'a',(A3)+
        CLR.L   D0                  ;CONVERT TO 32 BIT UNSIGNED
        MOVE.B  D1,D0               ;GET ROW POSITION
        BSR.S   @0002               ;ROUTINE TO COMPUTE OCTAL POSITIONING
        MOVE.B  #$80+'R',(A3)+
        CLR.L   D0                  ;CONVERT TO 32 BIT UNSIGNED
        MOVE.B  D2,D0               ;GET COLUMN POSITION
        BSR.S   @0002               ;ROUTINE TO COMPUTE OCTAL POSITIONING
        MOVE.B  #$80+'C',(A3)+
        CLR.B   (A3)+
        CLR.W   -(SP)
        RTR                         ;RETURN
*
@0002   ADDQ.L  #1,D0               ;BAISE ROW/COL BY 1
        DIVU.W  #100,D0             ;GET NUMBER OF 100S
        TST.W   D0
          BEQ.S @0003               ;NONE
        ADDI.B  #$80+'0',D0         ;OUTPUT NUMBER
        MOVE.B  D0,(A3)+
*
@0003   SWAP    D0                  ;GET 10S
        EXT.L   D0
        DIVU.W  #10,D0
        ADDI.B  #$80+'0',D0         ;OUTPUT 10S
        MOVE.B  D0,(A3)+
        SWAP    D0
        ADDI.B  #$80+'0',D0         ;OUTPUT 1S
        MOVE.B  D0,(A3)+
        RTS                         ;RETURN TO CALLER
```

Once the changes are made in MBIOS:SR the system must be regenerated using the following sequence:

```
xxDOS:GEN /ANS=1/CPSC=0,BASIC          ,BASIC if included
```

Test with xxLDGO and xxDOS.

Set up MTERM to send the clear screen sequence under the user mode. Then, you will have the special cursor positioning plus the normal clear screen commands. If it is a valid mode of operating, then finalize the system with the MMKBT utility.

Note: Other terminals may now have to be set up using the MTERM utility since the default is ANSI. If ANSI is not the default, drop the /CPSC=0 switch from the system generation command. The terminal with the higher column count must then be set using the MTERM utility for normal screen clear and BIOS cursor position.

6. **WORKAROUND -- NOT Operator in MASM.** The NOT operator token is not processed in QLINK. When doing arithmetic on XREFed labels, the assembler produces operator tokens in the object code output for the link to perform. The token produced by the NOT symbol (~) was left out of the QLINK list producing an error when INPUT to QLINK. So, don't use the NOT operator (~) on expressions with XREFs in them, but simulate it by adding and negating it. For example:

```
XREF    LABEL
MOVE.L  #~LABEL,D0
```

causes an error in the QLINKer, so change it to

```
MOVE.L  #-(LABEL+1),D0
```

7. **WORKAROUND -- MASM MOVEP Instruction Error.** The 68000 PDOS assembler MASM rev 3.0b or earlier generates an error on the MOVEP instruction with a 0 offset when ALT mode is enabled. To work around the problem, turn the ALT mode off around MOVEP instructions. Example:

```
OPT     NOALT
MOVEP.L 0(A1),D0
OPT     ALT
```

8. **WORKAROUND -- SYRAM Location in Custom Configurations**. If you make additions to the BIOS files, you must check the link map when regenerating the operating system to make sure that the end of operating system is less than the start of SYRAM.

   This means that you must define S$SRAM to be on a 2KB address bound and to be greater than the highest section address from the system generation. In the following example, S$SRAM must be moved to address $6800 since the link map indicates that the highest address is greater that $6000.

   From file xxDOS:MAP:

```
SECTION   BASE     LOWEST    HIGHEST
   E    00000800  00000800  00006020
   F    00000800  00001720  00006020 <-- Greater than $6000
```

Change 'DEFINE S$SRAM,$6000' to 'DEFINE S$SRAM,$6800' in file xxDOS:GEN and rerun xxDOS:GEN to build a new system file.

APPLICATIONS AND HINTS

1. **APPLICATION -- Pascal Task Data Passing.** The following PASCAL example illustrates how two PDOS tasks can coordinate the passing of data. This example comprises three files: HEADER:INC, SEND:PAS, and REC:PAS.

File HEADER:INC is used to define all common variables and the global mail box between the two tasks. This file is included when SEND:PAS and REC:PAS are compiled.

```
FILE=HEADER:INC

{**************************************************************
        PASCAL TASKING EXAMPLE OF GLOBAL MAIL BOX
        AND PDOS EVENT FLAG SYNC.

        THIS IS THE HEADER FILE FOR SHARED DATA DEFINITIONS
**************************************************************}

CONST
  EV_REC  = 64;         {PDOS EVENTS TO SYNC. ON}
  EV_SEND = 65;
  EV_STOP = 33;         {STOP EVENT TO EXIT ALL TASKS}

TYPE
  T_GLOBALS = RECORD            {SHARED VARIABLES BETWEEN PDOS TASKS}
    I : INTEGER;
    R : REAL;
  END;

VAR
  GLOBAL ORIGIN 16#70000 : T_GLOBALS;{SET SHARED VARS AT SOME FREE ADDRESS.
                                     WE WILL USE 70000 HEX.  YOU MAY HAVE
                                     TO USE SOME OTHER FREE ADDRESS}
{End of HEADER:INC}
```

SEND places data into a mail box (common memory area) and sets an event flag to allow the REC task to run. The program runs for 1Ø loops then sets an event flag that allows the REC task to exit. This program then exits.

```
FILE=SEND:PAS

{*************************************************************************
   PASCAL TASKING  EXAMPLE OF  GLOBAL MAIL BOX AND PDOS EVENT FLAG SYNCHRONI-
   ZATION

   THIS IS THE FIRST OF  TWO  PROGRAMS.   THIS  PROGRAM  SENDS  DATA  TO THE
   RECEIVER PROGRAM.  EVENT EV_PROG IS USED TO SYNCHRONIZE THE TASKS.
*************************************************************************}

PROGRAM SENDER;

{$F=HEADER:INC}                 {INCLUDE GLOBAL DEF FOR PROG}
VAR
  TEMP : INTEGER;                       {TEMP VAR}
  K : INTEGER;                  {FOR LOOP COUNTER}

{EXTERNAL PDOS PROCEDURES AND FUNCTIONS}

PROCEDURE XSEF(VAR T:INTEGER; EV:INTEGER);EXTERNAL;
PROCEDURE XSUI(VAR T:INTEGER; EV:INTEGER);EXTERNAL;

BEGIN
  XSEF(TEMP,EV_SEND);           {SET SEND EVENT TO RUN PROGRAM}
  XSEF(TEMP,-EV_REC);           {RESET REC EVENT TO WAIT}
  XSEF(TEMP,-EV_STOP);          {RESET STOP EVENT REC TASK}
  WITH GLOBAL DO
   FOR K:=1 TO 1Ø DO
    BEGIN
     XSUI(TEMP,EV_SEND);        {WAIT TILL OTHER PROGRAM IS READY}
     I:=K;                      {SEND GLOBAL MESSAGE INTEGER}
     R:=K/2;                    {SEND GLOBAL MESSAGE REAL}
     XSEF(TEMP,EV_REC);         {SET EVENT FLAG}
    END; {FOR}
  XSEF(TEMP,EV_STOP);           {STOP OTHER TASKS}
 END.
```

REC is used to receive the data after waiting for an event flag. It then prints the global data onto the screen and tests for an exit event flag. If the exit event flag is set, REC exits.

FILE= REC:PAS

```
{*************************************************************************
   PASCAL TASKING  EXAMPLE OF  GLOBAL MAIL BOX AND PDOS EVENT FLAG SYNCHRONI-
   ZATION

   THIS IS THE FIRST OF TWO PROGRAMS.  THIS PROGRAM RECEIVES AND  PRINTS DATA
   FROM THE  SENDER PROGRAM.  EVENT EV_PROG IS USED TO SYNCHRONIZE THE TASKS.
   THIS TASK RUNS UNTIL EV_STOP IS SET.
*************************************************************************}

PROGRAM RECEIVER;

{$F=HEADER:INC}         {INCLUDE GLOBAL DEF FOR PROG}
VAR
  TEMP : INTEGER;               {TEMP VAR}
  K : INTEGER;          {FOR LOOP COUNTER}

{EXTERNAL PDOS PROCEDURES AND FUNCTIONS}

PROCEDURE XSEF(VAR T:INTEGER; EV:INTEGER);EXTERNAL;
FUNCTION XTEF(EV:INTEGER):INTEGER;EXTERNAL;
PROCEDURE XSUI(VAR T:INTEGER; EV:INTEGER);EXTERNAL;

BEGIN
 WRITELN;
 REPEAT
  WITH GLOBAL DO
   BEGIN
    XSUI(TEMP,EV_REC);  {WAIT TILL SENDER HAS UPDATED MESSAGE}
    WRITELN('REC TASK: I=',I:1,' R=',R:5:2);
    XSEF(TEMP,EV_SEND); {SET SEND EVENT FLAG SO SENDER CAN RUN}
   END;
  UNTIL XTEF(EV_STOP)=1;        {RUN THIS TASK UNTIL EV_STOP IS SET}
END
```

First, compile and link SEND:PAS and REC:PAS:

```
>PASCAL SEND
>PASCAL REC
```

Next, run SEND as a background task and then execute REC. You should see the values for I and R displayed on the screen:

```
>CT SEND
*Task #2
>REC
REC TASK: I=1 R= 0.50
REC TASK: I=2 R= 1.00
REC TASK: I=3 R= 1.50
REC TASK: I=4 R= 2.00
REC TASK: I=5 R= 2.50
REC TASK: I=6 R= 3.00
REC TASK: I=7 R= 3.50
REC TASK: I=8 R= 4.00
REC TASK: I=9 R= 4.50
REC TASK: I=10 R= 5.00
>
```

The coordination of tasks and passing of data through a global memory area can easily be expanded to other variables and structures or converted to other languages.

2. **HINT -- FORTRAN PDOS Primitive Utilization.** Below is an example of the use of several integer function primitives under FORTRAN. In the example, XDEV sets up a delay of about 2 seconds on local event 128. XSUI suspends and waits for event (97) from port 1 and the timeout of event 128. XGCC receives characters from the port and resets the timeout. If no characters are input, the delayed event 128 aborts the character input. Note that XDEV, XSUI, and XGCC must be defined as INTEGER or they will not return the desired value.

Applications and Hints (cont.)

```
      PROGRAM TEST XDEV-XSUI-XGCC

      INTEGER ERROR,KEY,XGCC,XDEV,XSUI,J

10    CONTINUE
      ERROR = XDEV(200,128)
C       TYPE  'XDEV=',ERROR
      ERROR = XSUI(97*256+128)
C        TYPE  'XSUI=',ERROR
      IF (ERROR .EQ. 97) THEN
         KEY = XGCC(0)
C         TYPE 'KEY=',KEY
         J=J+1
         CALL XPCC(CHAR(KEY))
         IF (KEY .EQ. 13) GOTO 20
      ELSE IF (ERROR .EQ. -128) THEN
        GOTO 110
      ENDIF
      GOTO 10
20    CONTINUE                    ;take action for [CR]
      STOP 1
110   CONTINUE                    ;take action for timeout
      STOP 2
      END
```

3. **HINT -- Burning C Programs in ROM.** Variables initialized during the compilation generate code which locate the value in the ROM code. A variable may be modifiable or pre-initialized, but not both. If you desire both, declare two variables, one initialized and one not initialized. Then copy the initialized value to the uninitialized variable on startup routines.

4. **HINT -- PDOS Port Limitations.** PDOS versions 2.6F and 3.0 support up to 15 user console ports (SYRAM type ahead buffers). The number cannot be increased since user tasks cannot have unlimited input ports or events for input control. Polled input could extend the number of ports as desired but cannot be triggered by events under PDOS.

5. HINT -- **Force RTC Utilization and Change.** Force users can adjust the TPS (tics per second) on the RTC, by entering the FxBIOS:SR routine and changing the following code:

```
From    MOVE.B  #$0F,RAM(A2)    ;INTRPT EACH 1/100TH SEC
To      MOVE.B  #$F0,RAM(A2)    ;INTRPT EACH 1/1000TH SEC
```

The system must then be regenerated using the following command string:

```
>FxDOS:GEN /TPS=1000/RTCF=1
```

This will set the RTC clock to 1000 tics per second and initialize it as the system clock.

6. HINT -- **Force PIT Alternate Uses.** To use the parallel output of the PIT on your Force machine for other purposes besides the Centronics interface, you need to change the setup code in FxBIOSU:SR and eliminate the Centronics type of UART.

7. HINT -- **Zero RAM Disk Implementation.** If you don't want a RAM disk, setting the RZ-0 switch when assembling xxBIOS:SR causes the initialization code for the RAM disk to equal zero. This switch can used during system generation to define a zero size RAM disk on startup. This switch can be defined in MBIOS:SR or passed to the assembler on the command line.

```
>MASM S6BIOS:SR/RZ=0/IRD=0,OBJFILE,LISTFILE
```

8. HINT -- **Fine Tune Your PDOS Clock.** Would you like to fine tune your PDOS clock to be as accurate as your $5.00 watch? Read on.

CPU crystals do not run exactly at the posted speed. Many PDOS system TIC timers, from which the clock is derived, come from the processor's crystal. As such, the PDOS clock is notoriously wrong. It is not because we don't know how to count, but because the numbers printed on crystals are close together.

The following process takes a day or two, and can only adjust time-and-date clocks that run SLOW. If yours runs fast, you need to increase the TIMEC, timer counter load constant, in your xxBIOS:SR file.

Applications and Hints (cont.)

Once you have your timer running SLOW, you need to determine how many TICs per second (TPS) there are in your system. This number can be found in the *Installation and Systems Management Guide* for your system or in the xxBIOS:SR file. Force CPU-1 is used in this example. All the following code is found in the F1BIOS:SR file, and corresponding code for your system should be in the appropriate xxBIOS:SR file.

```
IFUDF   TPS     :TPS    EQU 100         ;TICS/SECOND
```

Now, suppose you set the PDOS clock to match your watch exactly at NOON, and the next day at NOON the PDOS clock is exactly 30 seconds slow, reading 11:59:30, then you have all the information needed to set CLKADJ. The PDOS clock lost 30 secs in exactly 24 hrs - 1,440 mins - 86,400 secs. So you need to add 1 TIC every 86400/3000, or 28.8, seconds. Now every second the B$LED BIOS routine is called from PDOS, in addition to blinking an LED, this routine does the clock adjustment. It does this by adding the CLKADJ value to a 32-bit counter every second, until it rolls over to zero, at which time it adds 1 to the TIC fine counter. The blink LED routine has the following code:

```
**************************************************
*       BLINK LED & ADJUST CLOCK
*
B$LED   MOVE.L  B_CLK(A0),D0        ;ADJUST CLOCK?
          BEQ.S @0002               ;N
        ADD.L   D0,BCLK.(A5)        ;Y, ADJUST COUNT, CARRY?
          BCC.S @0002               ;N
        ADDQ.W  #1,FCNT.(A5)        ;Y, UP COUNTER
*
@0002   RTS                         ;RETURN
```

This code will add a TIC whenever the 32-bit sum rolls over at 0, or put another way, when a number added to itself reaches 2^32. To find the CLKADJ number that will add 1 TIC every 28.8 seconds, use the following equation (thanks to Ward Horner):

```
CLKADJ = 2^32 * TPS * (secs lost) / (total measured secs)
```

In the example, CLKADJ - 2^32*100*30/86400, which equals 149130808.9 decimal, or $08E38E38 hex. You can now go into the debugger and alter the B_CLK value in the currently running BIOS table to try it immediately.

```
2>PB                            Enter debugger
0(5)                            SYram points to BIOS table
0000A000: 0000                  Table is at $00000AA0
0000A002: 0AA0
AA0: FFFF                       B_CLK is offset 8
00000AA2: FD6E
00000AA4: 5637
00000AA6: 0064
00000AA8: 0000 08E3[LF]         Enter upper word & [LF]
00000AAA: 0000 8E38[CR]         Enter lower word & [CR]
Q                               You're done
2>_
```

SYSGEN a new PDOS with the F1DOS:GEN file, temporarily setting CLKADJ. To generate a new PDOS system, using our value to predefine CLKADJ, type:

```
>F1DOS:GEN CLKADJ=149130809
...
```

Then, iterate on this value by setting the PDOS clock, now coarsely adjusted, to match your watch, and then seeing the 12 or 24 hour delta error. Then, adjust the value of 149130809 up or down by the newly calculated value, until the required accuracy is reached. Finally, to set this value once and for all, for this one CPU card at least, alter the F1BIOS:SR file line that sets the default CLKADJ value by replacing:

```
IFUDF   CLKADJ  :CLKADJ EQU 0           ;CLOCK ADJUST
```

with

```
CLKADJ  EQU     149130809
```

Make sure that you write your new PDOS out to your boot disk. Now your PDOS clock will have improved accuracy.

PDOS Realtime Operating System

Tips& TechnicalNotes

Vol. 1 No. 4
July 1, 1986

INTRODUCTION

Current Product Status

1. New PDOS Revision for 68020 Microprocessors
2. New Pascal Revision for both 68000 and 9900 PDOS
3. Floating Point Routines for Assembly Code

Warnings and Cautions

1. Caution - 102 Boot Responses
2. Caution - C Array Declarations
3. Caution - 68000 BASIC R3.0b - Negative Line Numbers
4. Warning - FORTRAN 77 R2.2 (M81:RL)
5. Warning - FORTRAN 77 R2.2 (68020)
6. Warning - FORTRAN 77 R2.2 (68000)
7. Warning - BASIC 3.0b - Calls to Assembly Programs

Fixes, Patches, and Workarounds

1. Workaround - BTST Instruction
2. Workaround - FORTRAN Byte and Word Constant Passing

Applications and Hints

1. Application - VME/10 Function Key Implementation
2. Application - Save Year in 58167 Battery Clocks
3. Application - Pack/Unpack Boolean Data in Pascal
4. Application - C Program Interrupt Trapping
5. Application - Protect BASIC Programs From Being Listed

CURRENT PRODUCT STATUS

1. A new release of PDOS (3.1) has been made for 68020 microprocessors. The release supports additional primitives and monitor commands as well as fully supporting the 68881 floating point chip.

2. PDOS Pascal has been upgraded to revision 3.0 for both 9900 and 68000 PDOS systems. Both the software and documentation have been significantly upgraded. If you wish to receive the Pascal revision upgrade, you must call or send a card to Eyring Research Institute, Inc., PDOS Customer Service, 1450 West 820 North, Provo, UT 84601, (801) 375-2434, Telex 882000. The Pascal upgrade is free of charge to warranteed Pascal customers but you must request it.

3. A special product disk has been made available which includes routines for single precision and double precision floating point calculations. Documentation assists the user in making the various floating point calls. This replaces the unsupported Fline commands from earlier versions of PDOS. Implementation includes 32 and 64 bit floating point routines. The floating point routines include the following:

   * FLOATING ADD/SUB
   * FLOATING COMPARE
   * FLOATING DIVIDE
   * INTEGER TO FLOAT
   * FLOATING MULTIPY
   * LOCAL SUPPORT -- NORMALIZE NUMBER
   * LOCAL SUPPORT -- FIX EXPONENTS FOR MUL/DIV /ETC
   * ROUND/TRUNCATE
   * TRANS FUNCTIONS ARCTAN
   * TRANS FUNCTIONS ERROR CONTROL
   * TRANS FUNCTIONS EXP
   * TRANS FUNCTIONS LOG (LN)
   * TRANS FUNCTIONS BREAK NUMBER INTO INTEGER AND FRACTION
   * TRANS FUNCTIONS FIX THE FPAC AND STACK FOR TRANS CAL
   * TRANS FUNCTIONS EVAL POLYNOMIAL (LOCAL FUNCTION)
   * TRANS FUNCTIONS SIN/COS
   * TRANS FUNCTIONS SQUARE ROOT
   * LOCAL SUPPORT -- SHIFT RIGHT 1 HEX DIGIT
   * LOCAL SUPPORT -- SCALE FLOATING POINT

WARNINGS AND CAUTIONS

1. CAUTION - 9900 users when responding to the boot device selection options, you must use the upper case 'Y' or the device will not be installed.

2. CAUTION - C array declarations specify an array size. The subscripts are 0 to size-1. For example,

   ```
   int a[5]
   ```

   will give the following array elements:

   ```
   a[0], a[1], a[2], a[3], a[4]
   ```

   C allows you to index outside the subscripts (i.e. a[5] or a[10]). You should note that other variables will be modified if you write to variables outside the limits of the declaration.

3. CAUTION - Documentation Change - 68000 BASIC 3.0b will not accept negative line numbers as documented on page 1-43 of the BASIC Reference Manual. The largest number accepted is 32767. Any higher number will be ignored.

4. WARNING - A bug has been noted in the ABSOFT FORTRAN (R2.2) M81:FL runtime library. A patch has been made by ABSOFT and will be made available when it is received. In the meantime, the user should use the F77:RL library. It will perform satisfactorily with little loss in efficiency.

5. WARNING - The FORTRAN 77 R2.2 implementation of the ATAN2(a1,a2) and ANINT(a1) give faulty results when run on the 68020 microprocessor with the 88881 floating point processor and the F77:RL floating point library. The division which should occur in the ATAN2 function does not occur and the result is the arctan of the first value. The ANINT function seems to pass the value of a nearby variable. These bugs have been reported to ABSOFT for their correction.

6. WARNING - The COS(0.0) function in FORTRAN 77 R2.2 on 68000 microprocessors gives a number slightly greater than 1.0 and, as a result, gives an error when you attempt to execute the ACOS function on the result of the COS(0.0) function. This is also true of the SIN function when the argument is PI. This problem seems to result from the rounding option used in these functions.

Warnings and Cautions (cont.)

7. WARNING - When chaining to an assembly program from BASIC 3.0b on the 68000 using the RUN command, register A5 does not point to SYRAM as is expected. Placing the XGML primitive at the beginning of the assembly program will initialize the registers to their proper values.

FIXES, PATCHES, AND WORKAROUNDS

1. WORKAROUND - 68000 MASM Rev. 3.0b rejects the immediate desgination addressing mode for the BTST instruction. The following macro will permit the instruction to be assembled properly:

```
*       BTST.B  D,#$06          ;TEST BIT IN REGISTER
*
BTSTX   MACRO
        DC.W    $013C+($&1-$D0)<<9,&Z
        ENDM
*
*       The instruction is included in the code using the
*       following call:
*
        BTSTX D1,$06
```

Proper assembly of the code will be implemented in a future release of MASM.

2. WORKAROUND - Some users of FORTRAN 77 R2.2 and earlier desire to pass constants as INTEGER*2 or INTEGER*1 format. This can be accomplished by multiplying the constant by 256^2 for WORDS and 256^3 for BYTES. This conversion is necessary because constants are defined as 32 bits and are always passed in INTEGER*4 format. Code that may work on other systems does not work on the 68000 microprocessors because of the hardware addressing mode.

   Since variables have TYPE, they can be defined as INTEGER*1, etc., and can be passed as such.

APPLICATIONS AND HINTS

1. APPLICATION - A number of PDOS VME/10 users would like to use the function keys in the PDOS editor MEDIT. The VME/10 keyboard is not a standard ASCII keyboard, and as a result, the keys must be interpreted by the BIOSU routines. A set of tables permit the keys to be decoded and send the ASCII code to the computer. You can customize your keyboard to fit your needs. Just replace the hex code for the key to the desired value in the VOBIOSU:SR file and then regenerate the system using VODOS:GEN. To test the change use the VOLDGO ,VODOS command. Once the keyboard is configured the way you want it, make the change permanent with MMKBT.

   CAUTION: Be sure not to change the relative location of the hex values in the table since this could affect more than the keys you are trying to change.

2. APPLICATION - On systems which contain the 58167 Real Time Clock and do not use this clock for the system clock, there is a patch which will permit the year to be saved in the RTC RAM.

   In MTIME:SR at the label CLCKTB, make the following change:

```
change:  DC.W   SYRS.,Ø,18,$18,99
to:      DC.W   SYRS.,19,18,$18,99
```

   This change saves the PDOS year in the RAM Hundreths and Tenths of Seconds area of the RTC when the MTIME B code is executed. Once stored, the year can be read into the PDOS system by the 'MTIME P' command.

   On FORCE CPU-1 systems, a change in the F1BIOS is required to prevent this RAM area from being overwritten during initialization.

```
change:     MOVEP.L DØ,RAM(A2)      ;SET RAM COMPARE ALWAYS
to:         MOVEP.W DØ,RAM+4(A2)
```

   Generate a new system and test it using the 'xxLDGO ,xxDOS' command. If it is what you want, make the boot permanent using MMKBT.

   To update the year from the PDOS clock on unattended systems, a task should periodically update the battery clock by running the 'MTIME B' utility.

3. APPLICATION - The following Pascal program shows two procedures, "PACKIT" and "UNPACKIT," which can be used to pack any boolean array into any other array type and to unpack the array back into a boolean array. These procedures can be modified so that various bit widths can be handled.

   You should also notice how PACKIT and UNPACKIT parameters are declared as pointers to an array. This allows the ADR function to be used by the calling procedure in passing arrays of any size.

```
TYPE
{$A=1}
   BYTE = -127..127;                             {A Byte}
   WORD = -32767..32767;
   TBOOLARY = ARRAY [1..10] OF BOOLEAN;          {Some Boolean array}
   TPACK_ARY = ARRAY [1..2 {DUMMY}] OF BYTE;     {Some dummy array}
   TPTR_PACK_ARY =^TPACK_ARY;                    {Pointer to dummy array}

VAR
  BOOLARY,
  BOOLARY2 : TBOOLARY;
  I : INTEGER;
  PACKWORD : INTEGER;


PROCEDURE PACKIT(DEST,SRC : TPTR_PACK_ARY; ELEMENTS:WORD);
 {Pack ELEMENTS number of the SRC Boolean array into the DEST bit array.}
 VAR
  SRC_INDEX : WORD;
  BYTE_INDEX : WORD;
  DEST_INDEX : WORD;
  BYTE_PTR : ^BYTE;
 BEGIN
  DEST_INDEX:=1;
  SRC_INDEX:=1;
  REPEAT
   BYTE_PTR:=ADR(DEST^[DEST_INDEX]);     {SET UP POINTER FOR FASTER CODE}
   BYTE_PTR^:=0;
   BYTE_INDEX:=1;
   REPEAT
      BYTE_PTR^:=BYTE_PTR^* 2;
      BYTE_PTR^:=BYTE_PTR^ + (SRC^[SRC_INDEX] AND 1);
      SRC_INDEX:=SRC_INDEX+1;
      BYTE_INDEX:=BYTE_INDEX+1;
   UNTIL (BYTE_INDEX>8);
   DEST_INDEX:=DEST_INDEX+1;
  UNTIL SRC_INDEX>ELEMENTS;
 END;
```

```
PROCEDURE UNPACKIT(DEST,SRC : TPTR_PACK_ARY; ELEMENTS:WORD);
 {UNPack ELEMENTS number of the SRC bit array into the DEST Boolean array.}
 VAR
  SRC_INDEX : WORD;
  DEST_INDEX : WORD;
  BIT_MASK : WORD;
 BEGIN
  DEST_INDEX:=1;
  SRC_INDEX:=1;
  REPEAT
   BIT_MASK:=16#80;
   REPEAT
    DEST^[DEST_INDEX]:=(SRC^[SRC_INDEX] AND BIT_MASK) DIV BIT_MASK;
    BIT_MASK :=BIT_MASK DIV 2;
    DEST_INDEX:=DEST_INDEX+1;
   UNTIL (BIT_MASK=0) OR (DEST_INDEX>ELEMENTS);
   SRC_INDEX:=SRC_INDEX+1;
  UNTIL DEST_INDEX>ELEMENTS;
 END;

{Main program to test the pack/unpack}
BEGIN
  BOOLARY[1]:=TRUE;
  BOOLARY[2]:=FALSE;
  BOOLARY[3]:=TRUE;
  BOOLARY[4]:=TRUE;
  BOOLARY[5]:=FALSE;
  BOOLARY[6]:=FALSE;
  BOOLARY[7]:=FALSE;
  BOOLARY[8]:=TRUE;
  BOOLARY[9]:=TRUE;
  BOOLARY[10]:=TRUE;
  PACKWORD:=-1;
  PACKIT(ADR(PACKWORD),ADR(BOOLARY),10);
  WRITELN;
  WRITELN('PACKWORD=',PACKWORD:-1);
  UNPACKIT(ADR(BOOLARY2),ADR(PACKWORD),10);
  WRITELN('COMP UNPACK TO NEW');

  FOR I:=1 TO 10 DO
   BEGIN
    WRITELN(BOOLARY[I]:10,BOOLARY2[I]:10);
   END;
END.
```

Applications and Hints (cont.)

4. APPLICATION - Some of you have asked how to write a C program that traps interrupts and reacts to them in some fashion. Unfortunately, it is not possible to do the job completely in C. When an interrupt arrives, you MUST save all registers, or the thing that you interrupted is probably going to be corrupted. When an interrupt occurs, the 68000 processor saves the current status register, switches to supervisor mode (and the supervisor stack), and saves the old status register and the old program counter on the stack. It then jumps indirectly through the interrupt vector. The routine called must exit via an RTE instruction to restore the old program counter and status register.

   These operations must be accomplished in assembly, since there is no straight-forward way to do them in C. The following code illustrates a stub that handles an interrupt by saving the current environment and calls a C subroutine to perform the majority of the function:

```
* PINT:SR -- ASSEMBLY INTERFACE

        XDEF .PINT
        XREF .INTSUB

.PINT   MOVE.W  #$2700,SR        ;DISABLE INTS
        MOVEM.L D0-A6,-(A7)      ;SAVE REGS
        JSR     .INTSUB          ;CALL C SUBROUTINE
*
PINT04  MOVEM.L (A7)+,D0-A6      ;RESTORE REGS
        RTE                      ;RETURN & HOPE
        END
```

   WARNING: In PDOS, the supervisor stack is not very big. If you intend to perform a large amount of work from the interrupt routine, you may need to save the old supervisor stack pointer and set up a new one that points to a larger stack area before calling the C subroutine. Naturally, you would then restore the old supervisor stack pointer when the C subroutine returns to the assembly code.

   To set up this stub so that it will be called on the appropriate interrupt, force load the interrupt vector to point to the stub. The interrupt vector here is at address $10C, which is the interrupt for the third port on a FORCE SIO card. You must determine the interrupt vector for your own interrupt.

Applications and Hints (cont.)

> When the interrupt occurs, the assembly routine PINT saves the registers on the stack, disables interrupts, and comes here. In this sample program, we will read the data from the interrupting port and output it to our own port. Only a limited number of PDOS functions are allowable during an interrupt. For instance, you may set or clear an event, set some global flag, or put data in a block of memory common to the interrupt code and some task. You may not do anything that requires knowledge by PDOS of a particular TCB because you do not know which TCB to use when you are in an interrupt. Thus, you may not use XPCC, XSTM, or XSWP.

```
/* C SUBROUTINE CALLED FROM INTERRUPT */
intsub()
{
        register char *input,*output;
        register char ch;
        input = 0xb00100;               /* BASE ADDRESS FOR PORT 5, SIO-1 */
        output = 0xf40000;              /* BASE ADDRESS FOR PORT 1, MPCC  */

#define rcvstat 1
#define rcvdata 3
#define xmtstat 9
#define xmtdata 11

        if (input[rcvstat] & 0x80){          /* check for data available*/
            ch = input[rcvdata];                /* read char from port */
            while (output[xmtstat] &0x80 == 0)  /* hang until ready */
                ;
            output[xmtdata] = ch;               /* write data to port */
            flag = ~flag;                       /* toggle flag for main */
        }
}
```

Once the interrupt vector is initialized, the C program can continue with non-critical functioning, or simply go into a loop executing an endless series of "XSWP" instructions. In this example, we will put asterisks to the screen until we read a character from the keyboard. At that point, we will restore the interrupt vector to its previous value and exit.

We will demonstrate that the main-line code can communicate with the interrupt routine by testing the variable 'flag' and printing either dots or asterisks. The interrupt routine will then toggle this flag each time it is called.

```
extern pint;            /* assembly language interrupt service routine */
int flag;               /* flag that communicates between main and intsub */

main()
{

/* INITIALIZATION CODE */

#define INTVEC *(long *) 0x10c

    long intsave;
    flag = 0;           /* initialize flag to false */
    intsave = INTVEC;   /* preserve old interrupt vector */
    INTVEC = &pint;     /* set interrupt vector to point to PINT */

/* OUTPUT ASTERISKS OR PERIODS WHILE WAITING FOR A KEYBOARD CHAR */

    while (xgcc() == -1) xpcc(flag ? '*' : '.');

/* TERMINATION */

    INTVEC = intsave;  /* restore vector */
    printf("\n that's all, folks!");
}
```

5. APPLICATION - To make your BASIC software so it cannot be listed, you should (1) use the NOESC command, (2) utilize an error trap to prevent the program from being interrupted, (3) purge all the code from memory on exit, and (4) save the file with the SAVEB command.

   The following example illustrates a method of protecting your code:

```
2>SAVE
**THIS PROGRAM IS PROTECTED AGAINST UNAUTHORIZED VIEWING**
ENTER YOUR NAME: DAVID
HELLO DAVID
ENTER YOUR PASSWORD: PASSWORD

STOP at line 75
LIST
 10  NOESC
 20  DIM NAME[3]
 30  ERROR 200
 40  PRINT '**THIS PROGRAM IS PROTECTED AGAINST UNAUTHORIZED VIEWING**'
 50  INPUT 'ENTER YOUR NAME: ';$NAME[0]
 60  PRINT 'HELLO ';$NAME[0]
 65  INPUT 'ENTER YOUR PASSWORD: ';$NAME[0]
 70  IF $NAME[0]='PASSWORD': ESCAPE : STOP   !THE CODE CAN BE VIEWED
 80  IF $NAME[0]<>'PASSWORD': GOTO 200
 100  I=KEY[0]  ! THIS REPRESENTS ANOTHER WAY TO PROVIDE AN ESCAPE
 110  IF I=1: GOTO 210  !^A PERMITS VIEWING ALSO
 120  GOTO 100
 200  PURGE 10 TO 120
 210  BYE
RUN
THIS PROGRAM IS PROTECTED AGAINST UNAUTHORIZED VIEWING
ENTER YOUR NAME: DAVID
HELLO DAVID
ENTER YOUR PASSWORD: TEST

2>EX
*READY
LIST
 200  PURGE 10 TO 120
 210  BYE
BYE
2>
```

PDOS Realtime Operating System

Tips & Technical Notes

Vol. 1 No. 5
September 1, 1986

INTRODUCTION

Product Status

Warnings and Cautions

Fixes, Patches, and Workarounds

Applications and Hints

PRODUCT STATUS

Following is a list of current revision levels of PDOS and supported languages. Those products preceded with an asterisk have just been updated. You may request an updated version of the products by contacting PDOS customer service and giving them your product serial number.

```
* 68000 Pascal Rev. 3.0a
* 9900 Pascal Rev. 3.0a
  68000/10 PDOS Rev. 3.0b
  68000/10 BASIC Rev. 3.0b
  68020 PDOS Rev. 3.1
  68020 BASIC Rev. 3.1
  68000 C Rev. 1.2c
* Absoft FORTRAN-77 2.2b
```

WARNINGS AND CAUTIONS

1. WARNING - Force WFC-1 disk initialization kills BINTB vectors. If you are using PDOS on any Force CPU with a WFC-1 disk controller, you will run into trouble if you try to add any interrupt vectors to the BINTB table. XDITW, the disk initialize routine for the WFC-1, reads Winchester drive header information into a temporary buffer. The 256-byte buffer starts at location $2FC, according to the FxBIOSW:SR code excerpted below:

```
    ...
       MOVE.B  D6,D0
       MOVEQ.L #$00,D1          ;SECTOR 0
       MOVEQ.L #$20,D2          ;READ COMMAND
       LEA.L   P$SASF-$104,A2   ;GET FAKE BUFFER ADDRESS
       BSR     WFCXX            ;DO A READ SECTOR 0
         BNE.S @070             ;READ ERROR, DO NOT INSTALL DRIVE
       LEA.L   DEFALTW(PC),A1   ;GET DEFAULT WFC PARAMS
       LEA.L   P$SASF-$104,A2   ;GET HEADER DATA AREA
 *
       @050    BSR.L   DOIT     ;MOVE DATA DOWN
    ...
```

Since P$SASF is at $0400, then A2 is set to $0400 - $0104 = $02FC. Though this would seem to work fine, if you try to add an interrupt routine to the BIOS and an entry in the BINTB for user vector #200, the interrupt routine address will be loaded by the kernel to vector location $0320, but XDITW will read disk data over it, from address $2FC to $3FB. As such, user vectors #191 through #254 will be destroyed.

The simplest solution is to use a 256 byte buffer somewhere else, for example, at location $0700. Do this by changing the following two instructions:

from ...

```
     LEA.L   P$SASF-$104,A2  ;GET FAKE BUFFER ADDRESS
      LEA.L   P$SASF-$104,A2  ;GET HEADER DATA AREA
```

to ...

```
      LEA.L   $0700,A2        ;GET FAKE BUFFER ADDRESS
      LEA.L   $0700,A2        ;GET HEADER DATA AREA
```

This change solves the problem. All future PDOS releases on the Force WFC-1 systems will incoporate the change.

Warnings and Cautions (cont.)

2. CAUTION - The MBACK utility on 68000 systems and BACKUP on 9900 systems causes the destination disk to become like the source disk. Don't make the mistake of trying to use this utility to transfer files from your floppy disks to a larger size PDOS disk. The results will be disastrous. Use MTRANS, TF, or CF on 68000 systems and TRANS and CF on 9900 systems. The backup utilities (MBACK and BACKUP) can be used to speed the transfer of files to floppy size disks on the Winchester and otherwise backing up entire disks onto blank disks. Move individual files or groups of files with the other utilities or monitor commands.

3. CAUTION - QLINK SRecord limits. If you are burning EPROMs using SRecords that are output from QLINK, BEWARE! The SRECORD command of QLINK has three parameters, [sadr], [eadr], [adr], for start address, end address, and SRecord base address. The start and end addresses are interpreted as absolute addresses, not buffer offsets or section values. Be careful when you enter the end address parameter, [eadr]. The last byte that QLINK outputs is from address [eadr] minus 1. Thus, to output 16k bytes (16384) into SRecords from QLINK, you need to enter:

```
SRECORD $0000,$4000,0      correct
SRECORD $0000,$3FFF,0      incorrect
```

The incorrect example only outputs 16383 bytes, leaving a byte of $FF in your EPROMs for you to find later on, when it doesn't work. So if you are breaking up a large file into SRecords to burn separately, use the following example as a guide:

```
SRECORD $0000,$4000,0          correct
SRECORD $4000,$8000,0
SRECORD $8000,$C000,0

SRECORD $0000,$3FFF,0          incorrect
SRECORD $4000,$7FFF,0
SRECORD $8000,$BFFF,0
```

The [eadr] parameter should actually be [eadr+1] and the SRECORD format SRECORD [sadr],[eadr+1]{,[adr]}.

Warnings and Cautions (cont.)

4. CAUTION - Although PDOS may reside at any location in memory, you may not relocate PDOS without adjusting the location of SYRAM. The xxLDGO utility will relocate SYRAM for you but MMKBT does not. It is necessary to adjust the SYRAM location (S$SRAM in xxDOS:GEN) and regenerate the system to properly relocate PDOS in memory.

5. NOTE - It is possible to set the length of a PASCAL string by assigning the value to the zero element as in:

   ```
   STRNG[Ø] := CHR(5);     {five character string}
   ```

   Be cautious when doing this, as some string operations manipulate the length as a 16-bit word. If any garbage is left in the high byte, it could cause problems. A good practice is to clear the upper byte of the length as follows:

   ```
   STRNG[-1] := CHR(Ø);
   ```

6. NOTE - Some 68020 users have experienced a problem with large library files when using the MLIB utility. Unless you include the [#sect] parameter, the size of MLIB:TMP is set at 100 sectors. If that size is too small to hold your library, the library setup will fail on PDOS error 56. See page 7-29 of the 68020 3.1 PDOS Reference Manual for details in using this utility.

7. DOCUMENTATION NOTE - The PDOS Reference Manual page 2-4 indicates that "Up to 32 independent tasks can reside in memory and share CPU cycles." But, by changing the 'NT' parameter in the MSYRAM module, PDOS can be configured to handle up to 128 tasks. This change will be made to future printings of the PDOS documentation.

FIXES, PATCHES AND WORKAROUNDS

1. FIX - An error in MFRMT:SR causes problems when you attempt to install multiple Winchester disks on a system. By making the following changes to MFRMT:SR, you can correct this problem:

Under label @0ZZ, change the following code:

```
     MOVE.L  A1,(A3)           ;SET DRIVE DATA DEFINITION
     BSR.L   RDHED             ;READ HEADER
       BEQ.S @0BB              ;OK
```

to:

```
     MOVE.L  A1,(A3)           ;SET DRIVE DATA DEFINITION
@RL  REG A1/A2/A3
     MOVEM.L @RL,-(A7)         ;SAVE SOME CRUCIAL REGISTERS
     BSR.L   RDHED             ;READ HEADER
     MOVEM.L (A7)+,@RL         ;RESTORE THEM
       BEQ.S @0BB              ;OK
```

Reassemble the source code and then follow the disk setup and format procedures in the installation guide.

2. WORKAROUND - The atan2() function in C does not work properly. To provide a fix which will permit the function to work, edit the file 'math.h' using the following command:

```
>MEDIT (math.h)[CR]
```

Change the line containing the atan2 declaration as follows:

```
/*double atan2();*/
#define atan2(x,y) atan((x)/(y))
```

Save the modified file 'math.h'. The atan2() function can then provide the correct result; i.e. atan2(1,1) gives 0.785398 radians. To convert this number to degrees, multiply the result by 180/pi where pi can be computed by pi = 4*atn(1).

3. WORKAROUND - The following example which deals with bit fields will not work properly with the current version (1.2c) of the C compiler. The bit field 'z.a' will not be assigned the value 1. To resolve this problem, do not use multiple variable expressions within the same assignment statement. This problem will be corrected in the next release of the C compiler.

```
struct ab {
  unsigned a : 1;
  unsigned b : 1;
};

main()
{
  struct ab z;

  z.a = z.b = 1;
}
```

4. WORKAROUND - A PDOS user has experienced a difficulty in using structures in C Rev. 1.2c. The assembly code which was generated would not permit the program to run unless a change was made before compiling.

   The following example and workaround illustrates the problem.

```
typedef struct { unsigned int a[9]; } b;
struct { b c[8192]; } *d;

main()
{
int x;
int i,j;

d = (unsigned int *)0x10;
i = 8191;
d = 5;

x = d->c[i].a[j];

}
```

The above code compiles to the following:

Continued . . .

```
          XDEF .MAIN
          SECTION 0
  .MAIN   EQU      *
          LINK A6,#-10
  *LINE 9
          MOVE.L #$10,.D
  *LINE 10
          MOVE #8191,-4(A6)
  *LINE 11
          MOVE.L #5,.D
  *LINE 13
          MOVE -4(A6),D0
          MULS #18,D0
          MOVE -6(A6),D1
          ASL #1,D1         <--------------These two lines
          EXT.L D1          <--------------should be swapped
          ADD.L D1,D0
          ADD.L .D,D0
          MOVE.L D0,A0
          MOVE (A0),-2(A6)
  L1      EQU      *
          UNLK A6
          RTS
          SECTION 1
          SECTION 2
          EVEN
          SECTION 1
          EVEN
          END
```

A workaround for this problem is to typecast the integer variable j to a long value.

```
x = d->c[i].a[(long)j];
```

The code will then be generated in the proper sequence.

5. WORKAROUND - The SGN function in 68020 BASIC 3.1 does not work properly on floating point numbers. To use this function at present, substitute SGN(INT(X)) for SGN(X).

This function will be fixed in a future release.

APPLICATIONS AND HINTS

1. HINT - Changing ASCII output on 9900 systems can be accomplished by changing the 9902 control register constant in PDOS and rebauding the port. On most systems, this value is stored at 086H and contains the value 6200H. There is a bias to 0A6H on PDOS 102 sytems. This sets the output of the 9902 to 7 bits even parity and 2 stop bits. The following table should allow you to select a configuration for your ASCII output string.

CONSTANT	CHARACTER LENGTH	PARITY STATUS	STOP BITS
4200H/5200H	7	NONE	2
4300H/5300H	8	NONE	2
6200H	7	EVEN	2
6300H	8	EVEN	2
7200H	7	ODD	2
7300H	8	ODD	2
8200H	7	NONE	1
8300H	8	NONE	1
A200H	7	EVEN	1
A300H	8	EVEN	1
B200H	7	ODD	1
B300H	8	ODD	1

This value can be changed using the 9900 BASIC statement:

```
MEMW(86H)=<constant in hex>
```

The MIAC utility can be used to view the 9902 control register constant and change it if desired.

If you wish to retain the new I/O conditions then save the change using the BFIX utility.

2. HINT - The trace window in the PDOS debugger defaults to the task dimensions when the task is created. If you should desire to execute code in another task in the trace mode, you need to expand the window to include the addresses to be traced. Since code outside of the trace window is not listed on the screen, a smaller trace window will permit you to check a selected block of code without having to step through code which is incidental to the problem being debugged. This can save considerable time in checking code which has many subroutine calls.

3. HINT - 9900 users who wish to customize the JEDY screen editor can do so by purchasing the source code. This code is part of the special product OWORD, a text runoff program, which is available for $250.

4. HINT - Following is a simplified discussion of using QLINK to separate your RAM and ROM when making run modules. The SECTIONs from the compilers and assembler give the key to separation:

PDOS kernel	- Section 15
PDOS BIOS	- Section 14
User code	- Section 0
User RAM	- Section 1
SYRAM RAM	- (offsets from A5)

The goal in this example is to put the PDOS kernel, the PDOS BIOS, and user code into ROM and then assign and group SYRAM, the user RAM, and the tasking RAM areas. It is assumed that you have run the first part of RUNGEN, which built the task 0, task 1, and task 2 object files. Also, MDUMMY:SR and MPDTST:SR files should have been assembled. This discussion only involves the QLINK part of RUNGEN.

Assume that:

SYRAM size	- $1000
Section 1 size	- $2340 (get info from compilers, etc.)
EPROM base addr	- $A0000

The ROM and RAM map then look like this:

	ROM at $A0000		RAM at $0000
	Task #2		TCB #2
	Task #1		TCB #1
	Task #0	$3800:	TCB #0
	R$TASK table	$2800:	SYRAM
	Kernel		Task #2 Sect 1
$A0000	BIOS		Task #1 Sect 1
		$0400:	Task #0 Sect 1
		$0000:	Vectors

The QLINK commands needed are:

```
BASE $A0000
SECTION 14,$A0000
GROUP 14,15,0
SECTION 1,$400
IGNORE 1
DEFINE B$SRAM,$03FC
DEFINE S$SRAM,$2800
INPUT xxBIOS:OBJ
...
SRECORD $A0000,Q$HE,0
OUTPUT #FILE:MX
END
QUIT
```

The BASE command sets the QLINK buffer to the address of EPROM. The SECTION 14 sets the BIOS code to link at the ROM address and the GROUP command combines the desired code sections. Next, set the base of the Section 1 RAM area, and ignore it, so that QLINK doesn't try to load this (RAM) into the (ROM) buffer.

Next, define where to store the SYRAM pointer (B$SRAM-$03FC), and define S$SRAM to be above the Section 1 stuff on an even $800 boundary. You are now ready to input all of the :OBJ files the RUNGEN utility tells you to (xxBIOS:OBJ must be the first). Now that all references are assigned the right location, you are ready to write the SRECORDs to burn, using the last address (+1) of section 14, Q$HE as the end address parameter.

Now all you have to do is send the FILE:MX file over to the ROM programmer, burn the ROMS, install them, and watch them work.

5. HINT - Using the PDOS 3.0 floating point routines from assembly -- or what do I do with all this 2.6 F-Line code I wrote?

Appendix F of the PDOS 3.0 manual describes the PDOS floating point module (MPDOSN:OBJ), which is part of the run module product but not a part of the standard PDOS package. The routines, however, are included in the code booted in for PDOS if you have PDOS BASIC (MPDOSBAS:OBJ). Under PDOS 2.6, you could access these routines using F-Line instructions (instruction words with the first nibble = $Fxxx which are commonly called Line-F instructions outside of the PDOS world). Thus, a user program only needed to enter the correct F-Line codes, and PDOS, with BASIC resident, would execute the requested floating point operation.

To improve the speed performance of BASIC, PDOS version 3.0 eliminated the F-line access to the routines, in favor of a direct BSR.L to a known location. This helped BASIC, but left assembly user programs without an address to stand on. One solution is to buy the run module package to get the MPDOSN:OBJ file to link with your application. Another solution was offered as a new product in Technical Notes, Vol. 1, No. 4, called "Floating Point Routines for Assembly Code". This product is essentially the PDOS Pascal Library for use with assembly. It includes transcendental functions, single and double precision, which were not included in the floating point module, but are nice. It still lacks decimal input and output conversion routines, and therefore has limited usefulness.

A third solution discussed here is that, if you have PDOS BASIC, you already have the Appendix F floating point routines in memory. You will learn how to find and use them.

The floating point code from the source file that created the MPDOSN:OBJ file is included at the very end of the MPDOSBAS:OBJ file, or right below SYRAM. First, find where the floating point routines are located within PDOS (get their base address in an address register), and then call them as offsets from the base address using the JSR instruction.

The following initialization code finds the routine base address and stores it in A4. Insert it into your assembly program that calls the floating point package:

```
*       ...                       ;ENTER FROM PDOS OR INIT CODE
        MOVEA.L A5,A4             ;POINT TO SYRAM
        LEA.L   -$4000(A4),A0     ;GET STOP LOOKING ADDRESS
*
@LOOP   SUBA.W  #2,A4             ;START LOOKING BACKWARDS
        CMPA.L  A0,A4             ;DONE?
          BHI.S @LOP2             ;N, KEEP LOOKING
        MOVE.L  #1999,D0          ;Y, REPORT ERROR 1999
        XERR                      ;AND EXIT
*
@LOP2   CMPI.L  #$262E0412,(A4) ;N, FLOAT FOUND?
          BNE.S @LOOP             ;N, KEEP LOOKING
        CMPI.L  #$2D50040E,-$2A6(A4) ;MAYBE, BEGINNING CORRECT?
          BNE.S @LOOP             ;N, KEEP LOOKING
        LEA.L   -$2A6(A4),A4      ;Y, POINT AT BEGINNING
*       ...                       ;REST OF INITIALIZE CODE
```

You now have the base address of the routines in A4. Next, define the offsets of the various routine entry points:

```
N$FABS  EQU     $0028             ;ABSOLUTE VALUE
N$FADD  EQU     $0056             ;ADD
N$FCLR  EQU     $0018             ;CLEAR
N$FDIV  EQU     $0210             ;DIVIDE
N$FELD  EQU     $0022             ;LOAD ERROR ADDRESS
N$FFLT  EQU     $00CE             ;FLOAT
N$FINV  EQU     $029C             ;INVERT
N$FLDD  EQU     $0000             ;LOAD FPAC
N$FMUL  EQU     $015A             ;MULTIPLY
N$FNEG  EQU     $0030             ;NEGATE
N$FNRM  EQU     $0108             ;NORMALIZE
N$FPST  EQU     $003E             ;READ STATUS
N$FSCL  EQU     $02BA             ;SCALE
N$FSRD  EQU     $000C             ;STORE FPAC
N$FSUB  EQU     $0046             ;SUBTRACT
```

You are now ready to make an assembly language call to the resident floating point package. The input and output formats are the same as defined in Appendix F (3.0) and Chapter 6 (2.6) of the PDOS Reference Manual, but instead of calling the routines with 'BSR.L N$Fxxx' (PDOS 3.0) or 'Fxxx.' (PDOS 2.6), call them with 'JSR.L N$Fxxx(A4)'. For example, to add the constant at LABEL to the FPAC, you would write:

```
        LEA.L   LABEL(PC),A0      ;GET ADDEND ADDRESS
        JSR.L   N$FADD(A4)        ;DO OPERATION
```

Looking back at the PDOS 2.6 Chapter 6, Floating Point Package, you will find that the F-line exception instruction for ADD was FADD., or $F004. Since the PDOS 3.0 assembler no longer predefines the F-Line mnemonics for the routine names, you could convert old PDOS 2.6 programs to use this method under PDOS 3.0 by defining some macros, named after the PDOS 2.6 F-Line calls. For example, for FADD. just define:

```
FADD.   MACRO
        JSR.L   N$FADD(A4)
        ENDM
```

Then the old reserved word, FADD., would assemble into the desired jump instruction, and not the old F-Line word. Of course, you must be sure that A4 is not destroyed in your program.

6. HINT - Method for extended I/O drivers: EXT:SR. PDOS I/O drivers must reside in the channel buffer, which is only 256 bytes long. The forward and backward file links take 4 bytes and the dedicated BRA.S table takes 6*2 more bytes, leaving only 240 bytes (-256-4-12) to work with. Many users have requested a method of expanding I/O drivers beyond this limit, by having code resident with PDOS.

   The following working example shows a multiple expanded driver file called EXT:SR. The idea is that you add as many large drivers as you want to the xxBIOS:SR file for your system, using the structure described below. Then to access them, you create some new disk resident drivers from the EXT:SR file, differentiating them by DNUM-0,1,2,...

   For example, to create files to access extended drivers #0 and #1 you would do the following:

```
0>SA DRV0,SY
0>MASM EXT:SR/DNUM=0,#DRV0
0>MSYFL DRV0,DRV0
0>SA DRV0,DR
0>SA DRV1,SY
0>MASM EXT:SR/DNUM=1,#DRV1
0>MSYFL DRV1,DRV1
0>SA DRV1,DR
0>_
```

Now there are two drivers, DRV0 and DRV1, to access each extended driver #0 and #1. This EXT:SR driver is a fixed length, which is important if you are going to store variables within the driver channel.

The only interesting call to EXT is OPEN, when it looks for the R$TASK table and a special EXT driver ID word ($5AA5). If you don't have any expanded driver code in the BIOS you booted, then EXT returns all calls with an error #99, but will not crash your system. If EXT finds the ID word, then it stores the address of the specified BRA.L instruction IN THE DRIVER at $10(A2). All the other entries to EXT just load up D0.L with the driver # (0,2,4,...) and an entry offset (0-open 4-close, 8-read,...) before branching (with an RTS) into the BIOS extended code entry point (stored in $10(A2)).

This keeps things all position independent, relocatable and re-entrant. Let's look at the EXT code before diving into the BIOS:

```
        TTL     EXT:SR - 68K PDOS 68K PDOS EXT DRIVER
*       EXT:SR          Ø6/27/86
******************************************************
*
*              66   888  K   K     PPPP  DDDD   000
*             6    8   8 K  K      P   P  D  D 0   0 S
*            6     8   8 K K       P   P  D  D 0   0 S
*            6666   888  KK        PPPP   D  D 0   0
*            6   6 8   8 K K       P      D  D 0   0
*            6   6 8   8 K  K      P      D  D 0   0 S
*             666   888  K   K     P     DDDD   000
*
*       EEEEE X   X TTTTT    DDDD  RRRR  III  V
*       E     X   X   T       D  D R   R  I   V   V E
*       E      X X    T       D  D R   R  I   V   V E
*       EEEE    X     T       D  D RRRR   I    V V  EE
*       E      X X    T       D  D R R    I    V V  E
*       E     X   X   T       D  D R  R   I     V   E
*       EEEEE X   X   T      DDDD  R   R III    V   EE
*
*=****************************************************
*       Eyring Research Inst.  Copyright 1983,1986.
*       ALL RIGHTS RESERVED.
*=
*=          Module Name: EXT
*=               Author: Richard Adams
*=      Revision History:
*=
*=      Ø6/27/86 3.Ø    Initial version of extended driver
```

```
*=
EXT     IDNT    3.0     68K PDOS EXT DRIVER
*=
*=*******************************************************
*
*       This driver is a general extended I/O driver, that
*       can be adapted for expanded driver code over the
*       252 byte limit.
*
*       D5.L = Character count (-1 = Line)
*       D7.W = Channel status
*       (A2) = Driver base + 4
*       (A3) = Memory buffer
*       (A4) = File slot
*       (A5) = SYSRAM
*       (A6) = Task TCB
*       (A7) = Return address
*
        IFUDF   DNUM    :DNUM   EQU     0       ;DEFAULT TO DRIVER #0
        PRINT   '  ** Extended driver # ',DNUM
        IFGT    DNUM-5
        PRINT   '  ** ERROR, Driver numbers only 0-5'
        ENDC
        PAGE
        SECTION 0

DEXT    DC.W    $A55A           ;DRIVER ID
DROP    BRA.S   OPEN            ; 2 OPEN
DRCL    BRA.S   CLOS            ; 4 CLOSE
DRRD    BRA.S   READ            ; 6 READ
DRWR    BRA.S   WRIT            ; 8 WRITE
DRPS    BRA.S   POSI            ;10 POSITION
        DC.L    0               ;Location of expanded code in BIOS
CODE    EQU     $10             ;CODE is channel offset of this saver
*
OPEN    ORI.W   #$8000,12(A4)   ;FILE ALTERED
        MOVEA.L (A5),A1         ;GET ADDR OF B$BIOS
        ADDA.L  (A1),A1         ;GET ADDRESS OF R$TASK TABLE
        CMPI.W  #$5AA5,-(A1)    ;IS ID THERE?
          BNE.S ERROR           ;N, DRIVER ERROR
        SUBQ.W  #4,A1           ;Y, POINT TO XTENDED CODE 'BRA.L'
        MOVE.L  A1,CODE(A2)     ;SAVE ENTRY
*
        MOVEQ.L #0,D0           ;0=open
```

```
*
*       CALL EXTENDED CODE WITH ENTRY OFFSET:
*         DØ.L = <minor offset> | <major offset>
*
*          Where <major offset> = Ø driver #Ø
*                               = 2 driver #1
*                               = 4 driver #2, etc.
*
*          Where <minor offset> = Ø open
*                               = 4 close
*                               = 8 read
*                               =12 write
*                               =16 position
*
CALL    MOVE.L  CODE(A2),-(A7)  ;GET ADDRESS
          BEQ.S EXTER           ;NO CODE, RETURN .NE.
        SWAP    DØ
        MOVE.W  #DNUM*2,DØ      ;GET DRIVER NUMBER OFFSET
        RTS                     ;GO TO CODE IN BIOS
*
CLOS    MOVEQ.L #4,DØ           ;4=close
        BRA.S   CALL
*
READ    MOVEQ.L #8,DØ           ;8=read
        BRA.S   CALL
*
WRIT    MOVEQ.L #12,DØ          ;12=write
        BRA.S   CALL
*
POSI    MOVEQ.L #16,DØ          ;16=position
        BRA.S   CALL
*
EXTER   ADDQ.W  #4,A7           ;POP CODE ADDRESS
*
ERROR   MOVEQ.L #99,DØ          ;if no extended driver code, err 99
        RTS
        END     DEXT
```

Note that from SYRAM (A5), you get the address of B$BIOS table and then calculate the address of R$TASK table. Place your $5AA5 EXT ID word right before R$TASK and the 'BRA.L XCODE' right before that.

To look at the xxBIOS:SR changes that let you add code there, let's get the example. The EXT example uses the TTA driver, adding it to the MVME117 V7BIOS:SR file. Just before the R$TASK table in the xxBIOS:SR file, you insert a BRA.L XCODE and an $5AA5 data word, as follows:

```
        ...
B$STRT  BRA.L   BSTRT           ;BOOT EPROM START
        DC.L    PDID            ;PDOS BOOT IDENTIFICATION
        DC.W    SYID            ;SYSTEM ID
B.SRAM  DC.L    S$SRAM          ;SYRAM ADDRESS
*
        BRA.L   XCODE           ;GO TO DRIVER CODE
        DC.W    $5AA5           ;EXTENDED DRIVER ID WORD
*
*****************************************************
*       TASK STARTUP TABLE (NON-RUN MODULE)
*
        IFEQ    RF
        XDEF    R$TASK
*
R$TASK  DC.B    1,U.1TYP,BR,%0000       ;PORT #1
        ...
```

Now following the BIOS interrupt routines, but preceding the INCLUDE MBIOS:SR command, insert the driver code. This could be done using an INCLUDE command, or even conditionally on an assembly flag. Define NDRV equal to the number of extended drivers in the xxBIOS (NDRV=1 in the example). You then have your major switchboard routine, XCODE, which checks the driver #, returning error 99 if it is too big. If D0.W is in range, then XCODE jumps to the particular driver code called by DRV0,DRV1, etc., with a JMP:

```
         ...
*************************************************************
*         EXTENDED DRIVER MAJOR ENTRY
*           IN:   D0.L = MINOR (0,4,8,12,16) | MAJOR (0,2,4,...)
*
NDRV     EQU     1                ;NUMBER OF DRIVERS RESIDENT
*
XCODE    CMPI.W  #NDRV*2,D0       ;IS MAJOR BRA.L IN TABLE?
           BLO.S @010             ;Y, GO TO IT
         MOVEQ.L #99,D0           ;N, THEN ILLEGAL
         RTS
*
@010     JMP     MAJOR(PC,D0.W)   ;GO TO DRIVER ENTRY
*
*        Main multiple driver switchboard table has each major
*        device entry is 4 bytes long, for a 'BRA.L DRVx' instruction.
*        The range is checked using NDRV, the number of drivers in BIOS.
*
MAJOR    BRA.L   DRV0             ;DRIVER #0 (TTA)
*        BRA.L   DRV1             ;DRIVER #1
*        BRA.L   DRV2             ;DRIVER #2
*        ...
*
```

In the example, only the standard TTA driver code has been added as DRVO. Since the driver entry points are now 0, 4, 8, 12, 16, you can have long jumps to the driver entry points, not limited to the 128 byte range. Another bonus is that for entries that are to return an error, such as read and position, you can handle the error RIGHT IN THE BRANCH TABLE! This is done by loading the error with a MOVEQ.L and RTS.

Variables within the driver (offset from A2) are very easy to define in the BIOS. Since you know the size of EXT:SR to be $4C, then by taking links into account you just use an OFFSET $50 directive, followed by DS.L, DS.W, and DS.B commands to yield the proper (A2) driver offsets. Remember to exit the OFFSET mode with a SECTION 14 command, for the linker:

```
*****************************************************
*       Extended Driver #0: TTA
*
*       Driver variables go here, starting at (A2) offset = $50
*       Use OFFSET and then return to section 14.
*
        OFFSET  $50             ;end of EXT driver code in buffer
PADR    DS.L    1               ;DC.L BASE ADR
FADR    DS.L    1               ;DC.L UART FLAGS ADDRESS

OUTE    DS.W    1               ;DC.W OUTPUT EVENT #
CCNT    DS.B    1               ;DC.B COLUMN COUNT
TYPE    DS.B    1               ;DC.B PORT TYPE
PUTC    DS.L    1               ;DC.L PUT CHAR ADDRESS FOR JSR
        SECTION 14              ;back to BIOS section
```

The next requirement is to reference in any external offsets or addresses:

```
*
*       Next define and XREF any needed offsets for SYRAM, etc.
*
BURT    EQU     $001E           ;BIOS UART TBL
        XREF    U2P$,UTYP.,UART.,F8BT.
```

Now, go to the specific driver code, which swaps DO to get the open, close, read, write, or position offset and branches into the fixed entry table to perform the driver function:

```
*       Here is the minor entry switchboard, with JMP offset in
*       upper word of D0.L.  Minor entry offsets are 0,4,8,$C,$10
*       for open, close, read, write and position.  This allows
*       errors in BRA.L table, with sequences like:
*
*               MOVEQ.L #ERR,D0
*               RTS
*
DRV0    SWAP    D0                  ;MINOR OFFSET IN D0.W LOWER
        JMP     DRV0TB(PC,D0.W) ;GO TO SPECIFIC MINOR ENTRY...
*
*       DRV0TB  BRA.L   OPEN
*               BRA.L   CLOS
*               BRA.L   READ
*               BRA.L   WRIT
*               BRA.L   POSIT
*
DRV0TB  BRA.L   OPEN                ;0=OPEN
*
        BRA.L   CLOS                ;4=CLOSE
*
        MOVEQ.L #80,D0              ;8=READ: ERROR 80, DRIVER ERROR
        RTS
*
        BRA.L   WRIT                ;12=WRITE
*
        MOVEQ.L #70,D0              ;16=POSITION: ERROR 70, POSITION ERR
        RTS
*
OPEN    ORI.W   #$8000,12(A4)       ;FILE ALTERED
        CLR.B   CCNT(A2)            ;CLEAR COUNTER
        CLR.W   D1                  ;D1=PORT #
        MOVE.B  U2P$(A6),D1         ;D1=PORT #
        MOVEQ.L #80,D3
        ADD.B   D1,D3
        MOVE.W  D3,OUTE(A2)         ;D3=OUTPUT EVENT #
        MOVE.B  UTYP.(A5,D1.W),D3 ;D3=UART TYPE
        MOVE.B  D3,TYPE(A2)         ;SAVE FOR FUTURE
        ADD.W   D3,D3               ;POINT TO DSR
        MOVEA.L (A5),A0
        ADDA.W  BURT(A0,D3.W),A0
        ADDQ.W  #2,A0               ;A0=PUTC ENTRY
        MOVE.L  A0,PUTC(A2)         ;SAVE PUTC ADR
        LSL.W   #2,D1               ;SAVE BASE ADR
        LEA.L   UART.(A5),A0
        MOVE.L  0(A0,D1.W),PADR(A2)
        LSR.W   #2,D1               ;SAVE FLAGS
        PEA     F8BT.(A5,D1.W)      ;PUSH POINTER TO FLAGS
        MOVE.L  (A7)+,FADR(A2)      ;SAVE PTR
        BRA.S   CLOS2
```

```
*
CLOS    MOVEQ.L #$0C,D0                 ;GET FF
        MOVEQ.L #1,D5                   ;DO 1 CHAR
        BRA.S   WRIT12                  ;OUT IT
*
CLOS2   CLR.W   D0                      ;RETURN .EQ.
        RTS
*
*******************************************************
*       WRITE CHARACTERS
*
WRIT    ORI.W   #$8000,12(A4)           ;N, ALTERED
*
WRIT02  MOVEQ.L #0,D0                   ;GET CHARACTER
        MOVE.B  (A3)+,D0                ;DONE?
          BNE.S WRIT04                  ;N
        TST.L   D5                      ;Y, WRITE LINE?
          BMI.S CLOS2                   ;Y, DONE
*
WRIT04  CMPI.B  #$08,D0                 ;BACKSPACE?
          BNE.S WRIT06                  ;N
        SUBQ.B  #1,CCNT(A2)             ;Y
*
WRIT06  CMPI.B  #$09,D0                 ;OK, TAB?
          BNE.S WRIT08                  ;N
        MOVEQ.L #' ',D0                 ;Y
        MOVE.B  CCNT(A2),D1             ;GET COUNTER
        LSL.B   #5,D1                   ;$CCC0 0000
        CMPI.B  #7<<5,D1                ;TAB BOUNDARY?
          BEQ.S WRIT08                  ;Y
        SUBQ.W  #1,A3                   ;N, DO AGAIN
        TST.L   D5                      ;WRITE LINE?
          BMI.S WRIT08                  ;Y
        ADDQ.L  #1,D5                   ;N, BACKUP
*
WRIT08  CMPI.B  #$0A,D0                 ;LF?
          BEQ.S WRIT16                  ;Y, IGNORE
        CMPI.B  #$0D,D0                 ;N, CR?
          BNE.S WRIT10                  ;N
        CLR.B   CCNT(A2)                ;Y, CLEAR CCNT
        MOVE.W  #$0A0D,D0               ;CHANGE TO CRLF
*
WRIT10  CMPI.B  #' ',D0                 ;CONTROL?
          BLT.S WRIT12                  ;Y
        ADDQ.B  #1,CCNT(A2)             ;N, UP COUNT
```

```
*
WRIT12  TST.B    TYPE(A2)         ;DEFINED TYPE?
          BEQ.S CLOS2             ;N, SKIP IT
        MOVE.L   OUTE(A2),D1      ;GET OUT EFVENT TO UPPER WORD OF D1
        MOVEA.L FADR(A2),AØ       ;GET PTR TO FLGS
        MOVE.B   (AØ),D1          ;TEST FLAG EACH TIME
        BTST.L   #Ø,D1            ;^S^Q CHECK?
          BEQ.S WRIT14            ;N
        TST.B    D1               ;Y, ^S STOP SET?
          BMI.S WRIT12            ;Y, WAIT HERE
*
WRIT14  MOVEA.L PADR(A2),AØ       ;UART BASE ADR
        MOVEA.L PUTC(A2),A1       ;POINT TO PUTC
        JSR      (A1)             ;CALL PUT CHAR
          BNE.S WRIT12            ;Y
        LSR.W    #8,DØ            ;N, 2 CHARS?
          BNE.S WRIT12            ;Y
*
WRIT16  SUBQ.L   #1,D5            ;DONE?
          BNE.S WRITØ2            ;N
        RTS                       ;Y, RETURN .EQ.
```

You would add other drivers here, calling them DRV1, DRV2, and so on. If you need more RAM storage than $100-$50 (176 bytes), then you would have to handle it separately. Also, you are limited to PDOS booting only up to 255 sectors, or less than 66k bytes for the BIOS, driver code and PDOS. This means that huge drivers must be accommodated differently. Now all that remains is to finish up by including MBIOS:SR.

```
*
        NOL
        PAGE
        INCLUDE MBIOS:SR
        END
```

7. HINT - To change the default word length, parity, and stop bits on a Force CPU-1 you may patch your system using the following method:

```
>PB
800,1000,7410W          search for first occurrence of 7410
 0F06                   address of occurrence
0F06 7410 7450          open this address and replace with 7450
Q                       exit the debugger

>BP $2002,9600          baud port with 7 bit, odd parity, 1 stop bit
```

Options for port communication without the patch are as follows:

```
>BP 2,9600              7 bits, even parity, 2 stop bits
>BP $2002,9600          7 bits, even parity, 1 stop bit
>BP $802,9600           8 bits, no parity,   2 stop bits
>BP $2802,9600          8 bits, even parity, 1 stop bit
```

Options for port communication with the patch installed:

```
>BP 2,9600              7 bits, odd parity,  2 stop bits
>BP $2002,9600          7 bits, odd parity,  1 stop bit
>BP $802,9600           8 bits, no parity,   1 stop bits
>BP $2802,9600          8 bits, odd parity,  1 stop bit
```

The patch may be saved with the MMKBT utility.

8. HINT - It is often desirable to do direct memory accessing from C, usually to memory-mapped I/O registers at a particular address.

You should already know that this type of code is machine dependent and should be isolated to a few small modules if portability is desired.

Should you desire to read/write 16 bits at a time to memory address 0xFF100200, you could define a pointer as follows:

```
int *p;
int i,j;

p = 0xFF100200;

i = *p;                /* read 16 bits from the address */
*p = j;                /* write 16 bits to the address */
```

If you will be making many references, you may want to declare the pointer to be a register variable to give quicker access and require less code.

For only one or two references to the address, you can simply declare the code in-line as follows:

```
i = *(int *) 0xFF100200;        /* read */
*(int *) 0xFF100200 = j;        /* write */
```

If you need to read/write a single byte, or 32 bits, you would declare the pointer above as follows:

```
char *p;                        /* one byte */
long *p;                        /* 32 bits */
```

or the in-line code:

```
c = *(char *) 0xFF100200;
c = *(long *) 0xFF100200;
```

9. HINT - Increasing performance on PDOS. As fast as PDOS generally is, there are some areas where it falls down. In particular, it seems as if compiling a program takes a long time. It turns out that an un-tuned file handling system is responsible for much of this apparent lack of speed. There are a few simple tricks you should know that will greatly increase PDOS performance without changing either your application or PDOS. Some of the tips are limited to 68000 PDOS but most of them can be used on either 9900 or 68000 PDOS systems.

PERFORMANCE TIP #1: PREDEFINE OUTPUT FILES TO THE LARGEST SIZE ANTICIPATED.

PDOS allows you to auto-create an output file by putting a pound sign ('#') in front of the file name. While this is generally convenient, it does cut back on your speed. An auto-created file is only defined with a size of one block, initially. When the file grows beyond that size, it must be extended. Each time the file is extended, PDOS goes back to the disk allocation bitmap at the beginning of the disk, marks off another sector, then goes back to the current sector to link in the new one. On a floppy disk you can hear the head "see-saw" back and forth as it extends a file sector by sector. You usually can't hear it on a hard disk but it happens nevertheless.

Also, when a file is extended in this fashion, the sectors are picked up on a first-come-first-served basis. The file will probably not be contiguous. Programmatic access is the same for non-contiguous files as it is for contiguous files, but there may be a large difference in performance -- especially in direct access files. This is because in order to read the Nth block, it is first necessary to read the preceding N-1 blocks as well. PDOS links blocks in a file together with a pair of links in each block. If the file is contiguous, PDOS knows where every block is on the disk by reading the disk directory. If the file is non-contiguous, however, PDOS must follow the chain of linked blocks to find the data.

So, for your own files and for the temporary files used by the compilers; PREDEFINE the output files. The C compiler uses the temporary files CTEMPx:SR1, CTEMPx:SR2, CTEMPx:O, and CTEMPx:L. The Pascal compiler uses PTEMPx:PIN, PTEMPx:PSR, and PTEMPx:POB. The 'x' is replaced by your task number -- 0, 1, 2, etc. The F77 compiler creates a series of files with the names filename:F1, filename:F2, etc., where 'filename' is the name of the file you are compiling. Unfortunately, you cannot predefine these files since the compiler defines the files as it goes and deletes them when it finishes. If you predefine the temporaries, the F77 compiler will not run. For the C and Pascal compilers and the assembler and linker, however, you can see a substantial speed increase by predefining the output files. Experience should show you how big these files should be. You should delete the old ones and create new ones of the maximum required size.

PERFORMANCE TIP #2: ORDER DIRECTORY TO PUT FREQUENTLY USED FILES FIRST.

The file directory on a PDOS disk is a simple list at the beginning of the disk. There are eight directory entries per sector. To look up a file on the disk, PDOS starts at the beginning of the directory and searches sequentially through the sectors until it finds the file or runs out of directory entries. Commonly used files, therefore, should be placed at the beginning of the directory. The system utility MORDIR allows you to arrange the disk directory in two different ways. First, it allows you to sort the directory alphabetically. Renaming a file does not change its order in the directory. To put the file 'XYZ:DAT' first in the disk directory, rename it to 'A', run MORDIR, and then rename 'A' to 'XYZ:DAT'. MORDIR also allows the disk level to be a sort key. You must put a '/L' after the disk number if you want this effect. To take advantage of this feature, you could put your most frequently used files in level 0, run MORDIR with the 'L' switch, and then (if necessary) rename them to the level you need them to be in. On a disk with a large number of files this can make a big difference in how fast you find a file.

PERFORMANCE TIP #3: KEEP DISK DIRECTORIES SMALL.

Along with the proper ordering of a disk directory, you should try to keep a disk directory fairly small. Generally, 100-200 files is about the most you should try to put on a single disk. Besides the overhead of looking through the long disk directory, you also have the problems of trying to keep track of what is what. Anyone will have trouble remembering the purpose of every file when the number of files gets up in the thousands. Backup unused files onto floppies and store them in a safe place. If you really must have thousands of files around, you have our sympathies and this suggestion -- back up the whole disk onto a set of floppies. Then, re-partition the large disk into a bunch of small ones using xxFRMT. Restore the floppies to the small (floppy-sized) disks. Typically, you can organize the files into functional groups i.e. by program, project, purpose, or something. With the 'SY' statement you can tell PDOS to search only those disks that are needed and to ignore the rest.

PERFORMANCE TIP #4: SET 'SY' PATH PROPERLY.

The 'SY' statement specifies which disks should be searched for filenames without an explicit disk reference. Up to four disks can be specified, with the numbers separated by commas or spaces. PDOS searches the disks in the order they are specified, from left to right. Thus, you should specify the SY command in the order that the files will most likely be found -- the most frequently referenced files should be on the first disk and the least frequently used ones should be on the last one. Likewise, fast devices ought to be searched before slow ones. Put the RAMdisk first, the floppy last, and a couple of hard disk partitions in the middle.

PERFORMANCE TIP #5: USE THE RAMDISK.

PDOS had RAMdisk long before it was popular on other operating systems. It allows you to free memory from tasking purposes and define it to be a high speed disk. You then initialize it and use it like any other disk. There are two obvious differences between the RAMdisk and other storage devices. The first difference is that the RAMdisk is volatile. Anything you place in RAMdisk should be copied to a regular disk as soon as possible, before the power fails and you lose it. The second difference is that since you don't have any moving parts in a RAMdisk, it runs MUCH faster.

Typical PDOS systems usually come with 256K of memory and 512K up to 1 or 2 megabytes is becoming increasingly common. The low cost of RAM makes it very easy to add a lot of memory to a system. Yet the system utilities on PDOS still only take up a few Kbytes and PDOS needs less than 50K. Most user programs will not take more than a few hundred Kbytes. The remaining memory can usually be given up to a RAMdisk.

When you compile a program in C, a lot of time is spent bringing in the different passes of the compiler from disk and referencing the different library files. If you can put the compiler phases and/or the libraries on a RAMdisk, you will easily see a doubling in your speed. Similarly, if you put a frequently accessed file on the RAMdisk, you will obviously see a speedup in your program. The PDOS programmers at Eyring try to use the RAMdisk as much as possible, especially for the intermediate files created by the various compilers.

* *

By implementing these performance tips, you should be able to increase the speed of PDOS on your system. If you have discovered other performance techniques that would be of general interest to PDOS programmers, please send them to Eyring or call the Hotline so that they can be included in the next issue of PDOS Tips and Technical Notes.

PDOS Realtime Operating System

Tips& TechnicalNotes

Vol. 2 No. 1
January 1, 1987

INTRODUCTION

Product Status

1. PDOS 3.2a release
2. C 5.0e release
3. FORTRAN 77 2.2d release
4. 9900 Pascal 3.0b release

Warnings and Cautions

1. BUG - Pointer overwrite in MEDIT
2. BUG - Rounding errors in 5.0 C floating point
3. BUG - MASM20 Bcc.X in 68020 mode
4. BUG - Pascal accessing byte data generates wrong code
5. BUG - Pascal 68k SQRT gives negative values near zero
6. WARNING - Use of task control block locations
7. WARNING - Pascal XSTM declaration doesn't work as documented
8. CAUTION - Assembly programs under FORTRAN
9. CAUTION - MDCOMP fails when comparing DRIVERS
10. CAUTION - Hardware effect on QLINK
11. CAUTION - BASIC file manager call - FILE 0
12. CAUTION - Use of LO command / XLDF primitive
13. CAUTION - Dynamic memory decay
14. CAUTION - Conflicting use of the monitor work buffer
15. NOTICE - 1.2 C vs 5.0 C floating point option

Fixes, Patches and Workarounds

1. PATCH - XDEV under 3.0 for run modules
2. FIX - XBFL primitive in C 1.2 lacks parameter
3. FIX - 9900 PASCAL run module
4. FIX - XCBC and XCBP in C primitive library

Applications and Hints

1. HINT - Nesting of procedure files
2. HINT - MASM/MASM20 variable definitions
3. APPLICATION - Interrupt service routines in PDOS
4. APPLICATION - Timing of benchmark routines
5. APPLICATION - Data files larger than PDOS disks
6. APPLICATION - 9900 Pascal run modules

PRODUCT STATUS

1. PDOS 3.2a is now available. It includes a number of new features including virtual ports, additional primitives and utilities, as well as updates to existing utilities.

   VIRTUAL PORTS - PDOS virtual ports (also referred to as "windows") allow selective switching of physical I/O ports to logical task ports. This means that a single terminal can dynamically switch between I/O ports which are assigned to different tasks or updated by a single task with multiple screen output. A screen image is maintained for all windowed ports so that when you switch from one port to another, your terminal is refreshed with the new screen.

   With PDOS virtual ports, the system acts as if there were more terminals than are actually on the system. Multiple tasks are accessible from one terminal. A high priority virtual port task maintains the screen buffers and handles screen refreshing and buffer printing. A special key sequence is used to switch from one virtual port to another. When a port selection is made, PDOS maps your keyboard to that port and the virtual port task clears and updates your display to reflect the current screen.

   To obtain a copy of PDOS 3.2 (68000 and 68020), call Susan Pitzak at Eyring or contact your distributor.

2. C version 5.0e is currently available. When PDOS-related products are revised, the lower case letters signify a patch or other minor fix and it is generally unnecessary to update your software. If the other numbers change, however, you might find it useful to obtain a copy of the new revision. (It will include new documentation and other substantial changes.) All users under a current maintenance/support agreement for C can obtain their copy of the latest release by requesting it. Others may obtain copies by updating their maintenance or ordering part number ER3550. Current cost of a new PDOS C license is $750.

3. FORTRAN 77 from ABSOFT is currently version 2.2d. Patch notices have been distributed to users with active license agreements. The 2.2d version will be distributed to all new customers.

4. 9900 Pascal 3.0b has been released. The new version includes coding to permit run module applications which was lacking in earlier versions.

WARNINGS AND CAUTIONS

1. BUG - In MEDIT 2.0 and earlier, if you overwrite the pointer in replace mode, the pointer cannot be found. As a result, cursor position and text pointers do not agree and text is written where it should not be written. This problem has been fixed in the version of MEDIT for the PDOS 3.2 release.

2. BUG - There are several rounding errors noted in the 'E' (IEEE) and the 'H' (68881) floating point librarys of 5.0 C. The 'E' library rounds incorrectly on output while the 'H' library truncates the result. Only the 'F' library rounds properly. The following example illustrates the 'E' library problem:

   ```
   double a = 123.004;
   printf("/n% %3.3lf",a);          /* prints 120.053 */
   printf("/n% %3.4lf",a);          /* prints 120.0090 */
   printf("/n% %3.5lf",a);          /* prints 120.00450 */
   printf("/n% %3.6lf",a);          /* prints 120.004050 */
   ```

   A patch for these difficulties will be provided in the near future.

3. BUG - In MASM20 (rev. 3.1), the Bcc.X in the 68020 mode does not assemble properly. This problem has been corrected in PDOS 3.2. For users of 3.1 PDOS, a patch or new distribution of the assembler may be obtained.

4. BUG - PDOS PASCAL 3.0 occasionally generates bad code when referencing functions that return a single byte as their result. As an example, consider the following program segment:

   ```
   TYPE
     TTYPE = 1..100;
       FUNCTION DUMMY:TTYPE;
       BEGIN
           DUMMY := 1;
       END;
   BEGIN
           IF DUMMY < DUMMY THEN WRITE(1) ELSE WRITE(2);
   END.
   ```

   Here, the function DUMMY is called twice, and the result from the first call compared to the result from the second. If DUMMY returned more than a byte (if TTYPE were the size of a 16-bit or 32-bit integer) then everything would work as expected. As it is, however, the compiler generates code that puts the stack pointer off by two, causing unpredictable results in the rest of the program.

(Warnings and Cautions continued)

```
*       IF DUMMY < DUMMY THEN WRITE(1) ELSE WRITE(2);

        SUBQ.W  #2,SP           ;ALLOCATE SPACE FOR FIRST RESULT
        PEA.L   (SP)
        BSR     .201            ;CALL FUNCTION

        SUBQ.W  #2,SP           ;ALLOCATE SPACE FOR SECOND RESULT
        PEA.L   (SP)
        BSR     .201            ;CALL FUNCTION

        MOVE.B  2(SP),D0        ;GET SECOND RESULT (LEFT ON STACK)
        CMP.B   (SP)+,D0        ;COMPARE TO FIRST RESULT (POPPED)

        BGE     .202            ;SECOND RESULT STILL ON STACK!

        ...
```

Presently, be warned that if you must have a function that returns a byte as the result, you should assign the result to a temporary variable, or use some other work-around to avoid calling two such functions as operands to a binary operator. The bug will be corrected in the next revision of Pascal.

5. BUG - The SQRT routine in Pascal 3.0a for the 68000 gives negative numbers below 0.0625 in both single and double precision modes. The problem is currently being investigated.

6. WARNING - Although the locations of the task control block are made available to the user, you must be cautious in using these locations. Many PDOS primitives use these locations to perform their functions and any location may change at any time as a result of these PDOS calls. Although the same task control block format has for the most part remained unchanged in PDOS revisions, this may not continue to be the case. As future improvements and changes are made to PDOS it may be necessary to modify the format significantly.

7. WARNING - The XSTM primitive does not work as documented in the Pascal 3.0 manual. Use the following declaration instead:

```
PROCEDURE XSTM(task:INTEGER; message:string);
```

(Warnings and Cautions continued)

8. CAUTION - The PDOS assembler generates a PDOS tagged object code which is accepted without difficulty by the FORTRAN linker F77L until you attempt to run the program. The result is usually an immediate program failure. It is necessary to convert assembly language subroutines to the SY format using the MSYFL or QLINK utilities before linking with the F77L linker.

9. CAUTION - MDCOMP will not compare drivers unless the driver attributes are removed or changed before the compare. Be sure to restore the attributes before attempting to use the driver.

10. CAUTION - A few C and Pascal users have reported 'illegal object tag' errors during the QLINK phase of creating a program. When we look at the library which holds the bad module, we cannot find anything wrong. It appears as if this problem is related to an idiosyncracy of a particular disk controller. One failure occured with the RWIN disk controller used on some VME-10 and 117 systems. The error goes away if we disable interrupts on the disk controller. If you have experienced this problem, you may try moving the appropriate files to RAM disk and linking there. Check your installation guide on procedures to disable the disk interrupt flag. We will continue to investigate this problem.

11. CAUTION - BASIC file manager call FILE 0 may not work properly on systems on which the BIOS routines suspend on events for timeouts. If a task which is locked attempts to suspend on an event, it cannot do it. The effect is that the task is immediately restarted and no timeout occurs. In this case, the FILE 0 call is made where the disk handler suspends on events, no time delay occurs, and a disk error is logged. BASIC programs should use the FILE 1 command to access files if difficulties are encountered.

    To prevent undesired access to shared files during disk reads and writes, use events to control access to the file routines. Locking and unlocking the open files will also assure that no other task can gain access to the file until it is released.

    This problem has been fixed for the 3.2 PDOS release.

(Warnings and Cautions continued)

12. CAUTION - The LO command which uses the XLDF primitive to load files into memory utilizes long word transfers to speed the process. As a result, the number of bytes loaded could be as many as three bytes greater than the number of bytes called for. Where programs are loaded, this presents no problem, but if data is being supplied to a program from an SY file, the space allocated for the data may not be sufficient and code may be overwritten. You must allocate sufficient memory to handle the possibility of the extra words transfer.

13. CAUTION - It has been noted that some dynamic memeory chips may hold their contents for several seconds. While this may help to avoid memory loss during a power glitch, it may also work against you. In one case, RAM disk memory survived a power-down/power-up cycle long enough to hold the A55A tag that PDOS checks to determine whether or not to initialize the RAM disk. The rest of the memory was garbage. As a result, the disk was not useable after the boot. If you must power down your system, it is recommended that you leave the power off for sufficient time to let the power supply go to zero before initializing power again.

14. CAUTION - Several PDOS primitives make use of a work area in the task control block called the monitor work buffer. These routines include numeric conversion routines (XCDB and XCBH); filename parsing by XFFN or any directory access; and the time and date conversion routines (XPAD, XRDT, XRTM, XUDT, XUTM, and XUAD).

    These routines use the monitor work buffer (MWB$(A6) or _tcbptr->_mwb) for scratch string space. To use the results of one call, you must use it before performing a second call. The second call will ovewrite the results of the first.

    It is unlikely that a programmer would try to print the date and time with the following assembly code:

    ```
    XRDT            ;get date string
    XRTM            ;get time string
    XPLC            ;print date???
    XPLC            ;print time
    ```

    In C, the equivalent code is not as clear and might appear as follows:

    ```
    printf("\nDate=%s; Time=%s",xrdt(),xrtm());
    ```

(Warnings and Cautions continued)

In this case, both the date and time conversions are performed, but the time result overwrites the date. When both strings are displayed, they are the same. The solution is to display the results of each function separately, or copy out the date into a buffer to preserve it while the time is being formatted. The following code in C is an example:

```
char dbuff[10],*strcpy(),*xrdt(),*xrtm();
printf("\nDate=%s; Time=%s",strcpy(dbuff,xrdt()),xrtm());
```

This example makes use of the fact that strcpy both copies the string and returns a pointer to the copy. The area 'dbuff' provides a temporary storage for the strings.

15. NOTICE - With the 1.2 C compiler, if you compiled to object code without specifying a floating point option, any floating point you had was by default generated in the FFP(option F) mode.

    With the 5.0 C compiler, the default floating point format is IEEE (option E). If you are in the habit of taking the default, be aware that the default has changed.

FIXES, PATCHES AND WORKAROUNDS

1. PATCH - XDEV under 3.0 for run modules. To apply the patch reported in Vol. 1 No. 3 of PDOS Tips and Technical Notes to run modules, do the following:

```
x>MEDIT MPDOSK1:OBJ[CR]                enter editor with MPDOSK1:OBJ
[CTRL-F]654E75[CR]                     find character string
[CTRL-A][CTRL-P][CTRL-F]5BC7[CR]   find next-place pointer-find string
[CTRL-\]V007C507004207[CTRL-W][CTRL-W][CR]
                                       del block-add string-write to file
V[ESC][CTRL-V]                         verify and exit editor
x>_
```

Since this patch is made to an object file, it will become part of any run module developed after the patch.

2. FIX - The build file directory list primitive in 1.2 C fails as a result of not passing all necessary variables. To correct this difficulty, enter the XLIB:SRC file and change the code under the XBFL primitive as follows:

```
change: .XBFL   MOVEM.L 4(A7),A1/A2      ;GET STRING POINTERS
 to:     .XBFL   MOVEM.L 4(A7),A1-A3      ;GET STRING POINTERS
```

It will be necessary to separate the library modules, recompile, and link them into a new library file using MLIBGEN, or use the new MLIB utility distributed with PDOS 3.1 and later to change the single library routine following compilation.

This problem has been corrected in 'C' 5.0 which is now available.

3. FIX - 9900 Pascal Rev 3.0a files PMAIN:OBJ and EOPG:OBJ are not correct for run module generation. Rev 3.0b corrects this problem and is available to those who need to use Pascal in run modules.

4. FIX - The primitives XCBC and XCBP require a modification to assure proper operation. The following changes should be made in the XLIB:SRC library sources and the library file reconstructed. In both routines, make the following changes thus assuring that the D0 register returns a 0 when no break character is encountered rather than the old D0 value.

```
change:                                  to:
                MOVE.L  D1,D0                          @0099   MOVE.L  D1,D0
        @0099   RTS                                            RTS
                END                                            END
```

APPLICATIONS AND HINTS

1. HINT - Nesting of procedure files is accomplished by placing the file ID in the variable ACI$ in the task control block. There is currently space for two file IDs allowing a third program to execute while nesting two procedure files. A third procedure file will not execute.

   To expand the nesting, you could write a program which pops the last ID and saves it. When the nested procedure is complete, a second program should push the ID back to ACI$ (assigned console inputs). Using this technique, you can nest procedure files as deep as you wish.

2. HINT - When using the PDOS assembler MASM or MASM20, it is important to "well-define" all variables before they are used in a program. If this is not done, an error will result in the first pass of the assembler. If the definition is later processed, there may be no error reported in the second pass. You may not assume that the variable has been properly assigned or used. If you were to test the error value LEN$, it would contain a non-zero value, indicating an error occurred. It does not matter whether the error occurred in the first or second pass.

3. APPLICATION - Interrupt service routines for new cards in a PDOS system are usually added to the xxBIOS:SR file, with a new entry in the BINTB table. Note the examples in xxBIOS:SR on your system. The BINTB table indicates all interrupt vectors already assigned for your system.

   Some general rules are:

   A. Be sure that the interrupt acknowledge daisy-chain is complete across the backplane on VMEbus systems.

   B. For response-critical applications, select an interrupt level higher than the system clock interrupt.

   C. Save all registers used in the routine upon entry and restore them before exiting.

   D. Do not assume any registers are preset or passed from a task.

   E. Since the system stack is not infinite, consider the possibility of interrupting during another task or during the SWAP routine. Adjust your code accordingly.

   F. Avoid wait loops in interrupt service routines (ISR).

(Applications and Hints continued)

G. Avoid using PDOS primitives in the ISR.

H. Preferably set an event and return with either an RTE instruction or an XRTE primitive.

The best way to handle device interrupts from a task is to set the device to interrupt and then suspend the task on both a timeout local event and the event (EVNT) associated with the ISR. Be sure EVNT is reset before suspending or you will come right back before the interrupt. The ISR should set EVNT directly in SYRAM with code similar to the following:

```
         XREF     EVTB.
*
ISR      MOVE.L   A5,-(A7)                ;SAVE REGISTER
         MOVE.L   B$SRAM,A5               ;GET SYRAM POINTER
         BSET     #~EVNT,EVNT/8+EVTB.(A5) ;SET EVENT IN SYRAM TABLE
         MOVEA.L  (A7)+,A5                ;RESTORE REGISTER
         XRTE                             ;RETURN AND SWAP TO TASK
```

The '~' on EVNT simply converts PDOS event numbers (where 0 is most significant) to 68000 bit numbers (where 7 is most significant).

Dividing EVNT by 8 yields the byte index of event EVNT in the event bit table of SYRAM. The XRTE primitive executes an RTE instruction after setting a flag for the PDOS swap routine to execute a swap as soon as possible.

If faster interrupt response is needed or if some immediate calculation of data is required, then you need to insert more code at the ISR itself. Remember not to block interrupts for too long, unless absolutely necessary.

(For systems using the Force WFC-1 card, see the warning in PDOS Tips and Technical Notes Vol. 1, No. 5.)

4. APPLICATION - UTILITY FOR TIMING BENCHMARK PROGRAMS

There is often a need to accurately time various programs for benchmark comparisons. The following example will help you in performing these timings:

Using MEDIT, enter and save the programs listed following as files on your disk in START:SR and TEND:SR.

Assemble them to create assembly language programs.

(Applications and Hints continued)

```
>MASM START:SR,#START
>MASM TEND:SR,#TEND
```

Timings of programs can be made by using the following command line at the PDOS prompt:

```
>START.PROGRAM.TEND            ;run START,PROGRAM, and TEND
```

To determine time for execution of START and TEND, execute the following command:

```
>START.TEND

*START:SR
*********************************************************
*       START OF TEST
*********************************************************
        OPT PDOS
START   XPMC    MES03           ;'START'
        XGML
        MOVEA.L MAIL.(A5),A2    ;GET MAIL ARRAY ADDRESS
        MOVE.L  TICS.(A5),12(A2)        ;SAVE TICS IN MAIL ARRAY
        XEXT
MES03   DC.B    $0A,$0D,'START....',0

        END START


*TEND:SR
*********************************************************
*       END OF TEST
*********************************************************
        OPT PDOS
*
TEND    XGML
        MOVEA.L 4(A5),A2        ;GET MAIL ARRAY ADDRESS
        MOVE.L  12(A2),D7               ;GET START TICS
*
        SUB.L   TICS.(A5),D7    ;GET TIME
        NEG.L   D7
        MOVE.L  D7,D6           ;SAVE UPPER
        SWAP    D6
        MULU.W  #100,D7         ;MULU LOWER GUY
        MULU.W  #100,D6         ;MULU UPPER
        SWAP    D6              ;UPPER*65536
        ADD.L   D6,D7
        XRTM                    ;GET TIC PER SECOND AT 10(A1).W
        MOVE.W  10(A1),D0
        MOVE.L  D7,D1
        BSR.S   FDIV            ;DIVIDE D1 BY DO
```

(Applications and Hints continued)

```
        LSL.W   #1,D2           ;GET REMAINDER * 2
        CMP.W   10(A1),D2       ;ROUND IT UP?
          BLO.S TEND2           ;N
        ADDQ.L  #1,D1           ;Y, ROUND UP TO NEXT TIC
*
TEND2   MOVEQ.L #100,D0
        BSR.S   FDIV            ;GET HUNDREDS
        XPCL
        XCBM    MES04           ;'TIME='
        XPLC                    ;OUTPUT WHOLE #
        MOVEQ.L #'.',D0
        XPCC
        MOVE.L  D2,D1           ;GET REMAINDER
        ADDI.W  #100,D1
        XCBD
        ADDQ.W  #1,A1
        XPLC                    ;OUTPUT FRACTION
        XPMC    MES05
        XEXT
*
**************************************************
*       DIVIDE D0.W INTO D1.L (D2.W=REMAINDER)
*
FDIV    MOVEQ.L #0,D2           ;CLEAR REMAINDER
        MOVEQ.L #32-1,D3        ;GET COUNT
*
@IFD2   ADD.L   D1,D1           ;SHIFT LEFT D2,D1
        ADDX.W  D2,D2
        CMP.W   D0,D2           ;D2 <= D0?
          BLO.S @IFD4           ;Y
        SUB.W   D0,D2           ;N, D2 = D2 - D0
        ADDQ.L  #1,D1           ;ENTER BIT
*
@IFD4   DBRA    D3,@IFD2        ;DONE?
        RTS                     ;RETURN
*
MES04   DC.B    'END.  TIME=',0
MES05   DC.B    ' SECONDS',0
        END TEND
```

5. APPLICATION - There have been a number of requests for the means of accessing files larger than the PDOS size limit under the PDOS file manager. A simple technique which keeps track of the maximum size of each file and selects a different logical PDOS disk can be used. By defining files to hold a number of record sets per disk, the program can direct the output or find the data based on an index into the file on the appropriate disk.

(Applications and Hints continued)

The following example written in PDOS BASIC creates three files on three separate logical PDOS disks and initializes the files with data. The program then accesses this data randomly by calculating the record to be accessed and the disk on which it is located. An approach similar to this can be used in all languages.

```
1  REM  PROGRAM TO CREATE MULTI-DISK DATA FILE AND ACCESS
2  REM  DATA FROM THAT FILE RANDOMLY
5  M=0
10  DIM FILE[5],F[5]
20  $FILE[0]='#DTEMP/'                !INITIALIZE FILENAME
30  FOR J=55 TO 57                    !DO DISKS 7-9
40   $F[0]=$FILE[0]%J                 !CONCATENATE DISK # TO FILENAME
45   OPEN $F[0],ID                    !OPEN FILE ON DISK
50   FOR I=(J-55)*100 TO (J-54)*100-1  !0-99,100-199,200-299
60    FILE 1,ID;2,I,I*I,I*I*I         !WRITE #, #^2, #^3 TO FILE
70   NEXT I
80   CLOSE ID                         !CLOSE EACH FILE
90  NEXT J                            !REPEAT FOR EACH DISK
100  I=INT[RND*300]                   !GENERATE RANDOM NUMBER
110  IF I<100: J=55: GOTO 130         !CHECK DISK NUMBER IS ON
115  IF I>199: J=57: GOTO 130
120  J=56
130  $F[0]=$FILE[0]%J                 !SET UP FILENAME AND DISK
140  ROPEN $F[0],ID                   !OPEN FILE ON APPROPRIATE DISK
145  K=I-(J-55)*100                   !CALCULATE POSITION IN FILE
150  FILE 1,ID;4,3*8,K,0              !POSITION TO DATA
160  FILE 3,J,K,L                     !READ DATA AND PRINT IT
170  IF I<>J: PRINT 'ENTRY ';I;' READ AS ';J,K,L
180  PRINT I,J,K,L
185  CLOSE ID
190  REM  WAIT 112
195  M=M+1
196  IF M=100: UNIT 1:BYE
200  GOTO 100
```

6. APPLICATION - With release 3.0b of the 9900 Pascal compiler, run module generation has been simplified. The following is a short example of run module generation on a 101 9900 PDOS system. You will need the following in order to produce run modules on your computer:

<u>Software Products:</u>

9900 Run modules Rev 2.4d (Part number: 3411)
9900 Pascal Compiler Rev 3.0b (Part number: 3430)

Upgraded PDOS Utilities:

LINK Rev 2.4d (supplied on Pascal Rev 3.0b diskette - part number 3430)

You should also become familiar with chapter 12 of the 9900 PDOS Reference Manual. If you have never done run modules, you may wish to try some of the assembly examples that are in chapter 12 first.

After you have become familiar with the PDOS run module examples in chapter 12, you are then ready to build a Pascal run module. First you will need to write your Pascal programs. You will need to add an XBCP call at the start of any Pascal task that will be using character I/O so that the port, baud rate and CRU base are properly selected. The following is a short program that you may wish to use to build your first run module.

```
PROGRAM PASMOD;

VAR
  I : INTEGER;

PROCEDURE XBCP(PORT,BAUD,PTYPE,BASE:INTEGER);EXTERNAL;

BEGIN
  XBCP(1,0,0,16#80);    {MUST BAUD PORT}
  REPEAT
   WRITELN;
   WRITE('ENTER A NUMBER: ');READLN(I);
   WRITELN('NUMBER * 100 =',I*100);
  UNTIL FALSE;
END.
```

Save the file in PASMOD:PAS. Then you must compile and debug the program.

```
.PASCAL PASMOD

.PASCAL1 PASMOD:PAS/5,#PTEMP0:PIN/5
9900 PDOS Pascal  R3.0a 25-Jul-86
 Copyright 1984-1986 ERI
SRC=PASMOD:PAS/5
INT=#PTEMP0:PIN/5
LST=
ERR=
```

(Applications and Hints continued)

```
<0>..............
LINES:15
ERRORS:0
GLOBALS=2 BYTES

.PASCAL2 PTEMP0:PIN/5,#PTEMP0:PSR/5
9900 PDOS Pascal Code Generation  R3.0a 25-Jul-86
 Copyright 1984-1986 ERI
INPUT=PTEMP0:PIN/5
OUTPUT=#PTEMP0:PSR/5

.PASM PTEMP0:PSR/5,#PASMOD:POB/5
PASM R2.6b
SRCE=PTEMP0:PSR/5
OBJ=#PASMOD:POB/5
LIST=
ERR=
XREF=

END OF PASS 1
0 DIAGNOSTICS
END OF PASS 2
0 DIAGNOSTICS
.PLINK
LINKER R2.4a
*0,#PASMOD/5
*12,2
WAS >0000
*1,PMAIN:OBJ
START TAG = >0000
*1,PASMOD:POB/5
*13,PLIB1:LIB
*13,PLIBS:LIB
*13,PLIBX:LIB
 1,TXBCP:OBJ
*13,PLIBF:LIB
*13,PLIBIO:LIB
 1,TPRDINT:OBJ
 1,TPGETCH:OBJ
 1,TPIOOK:OBJ
 1,TPRDLNF:OBJ
 1,TPWRINT:OBJ
 1,TPPUTCH:OBJ
 1,TPWRLNF:OBJ
 1,TPWRSTF:OBJ
```

(Applications and Hints continued)

```
*13,PLIB:LIB
 1,TPEND:OBJ
 1,TPERROR:OBJ
 1,TPPARL:OBJ
 1,TPDISP:OBJ
 1,TPNEW:OBJ
*1,EOPG:OBJ
*2
UNDEFINED DEF ENTRIES: NONE
*3
MULTIPLY DEFINED DEF ENTRIES: NONE
*6
START TAG = >0000
*7
.
```

After you have linked the Pascal modules together, try the program under PDOS.

```
.PASMOD
ENTER A NUMBER: 1
NUMBER * 100 =          100

ENTER A NUMBER: 2
NUMBER * 100 =          200

ENTER A NUMBER: [ESC]
.
```

You are now ready to try this program as a run module. File R$MODC:SR must be edited with the configuration parameters that you need for this run module. This includes defining the number of tasks, the port definitions, the clear screen and position cursor commands along with other parameters that are described on page 12-6 of the 9900 PDOS Reference Manual. It is recommended that you edit a copy of file R$MODC:SR and save the changes in another name. For this example, call the changed file RUNMODC:SR. You need to assemble this file.

```
.PASM RUNMODC:SR,#RUNMODC
PASM R2.6b
SRCE=RUNMODC:SR
OBJ=#RUNMODC
LIST=
ERR=
XREF=
```

(Applications and Hints continued)

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

You are now ready to link the run modules. This example will show how the new link command "16 -- Load a Pascal program" is used. You should note that this command is very similar to the "11 command -- Load a Basic program".

```
  .LINK
  LINKER R2.4b                  LINK MUST BE 2.4b
  *9,>4000                      Set the DSEG (RAM) base address
  WAS >0000
  *12,0                         Ignore DSEG code
  WAS >0000
  *0,#RUNMOD                    Select output file
  *1,R1$MODA                    Load PDOS run modules
  START TAG = >0000
  *1,RUNMODC                    Load configuration module
  START TAG = >09FC
  *16,PASMOD,16,1               Load the Pascal program with 16kb RAM
  START TAG = >0AA6              on Port 1
  *2
  UNDEFINED DEF ENTRIES:        You will see undefined entries

 BASIC  >0000  CVBD   >0000  CVBI   >0000  EFLG   >0000  EVFX   >0000
 FAD      >0000  FDD    >0000  FLDD   >0000  FMD    >0000  FOPS   >0000
 FSCL   >0000  FSD    >0000  FSRD   >0000  GOSBE  >0000  SY$IN  >0000
 TYPV1  >0000  TYPV2  >0000
  *3                            Check for multiply-defined defs
  MULTIPLY DEFINED DEF ENTRIES: NONE
  *4,#RUNMOD:MAP                Output a map
  *6                            Output a start tag
  START TAG = >0000
  *7                            Exit back to PDOS
  .
```

You should note that undefined entries are common. However, you should not see undefines of the format R$PMxx R$PTxx R$DBxx or R$DExx (where xx is a two digit number). These undefines indicate that the module R$MODC:SR has more tasks defined than you loaded. This will cause your run module not to work.

You can next test your run module by using the LOGO program.

```
.LOGO
LOGO R2.4
*1,RUNMOD
LOADING.....
IDT='R$MA2.4c'
IDT='R$MC2.4c'
IDT='3.0bMAIN'
IDT='P3.0a    '
IDT='2.7TXBCP'
IDT='1.0RDINT'
IDT='2.7GETCH'
IDT='2.7IOOK '
IDT='2.7RDLNF'
IDT='1.0WRINT'
IDT='2.7PUTCH'
IDT='1.0WRLNF'
IDT='2.0WRSTF'
IDT='2.7PEND '
IDT='3.0PERR '
IDT='2.7PARL '
IDT='2.7DISP '
IDT='2.7NEW  '
IDT='3.0bEOPG'
ENTRY ADDRESS=>0000
*0                      Execute it!
GO !!!

ENTER A NUMBER: 1       It worked!!
NUMBER * 100 =          100

ENTER A NUMBER: 2
NUMBER * 100 =          200

ENTER A NUMBER: [ESC]   Exit
PDOS ERR=81             Error because no monitor
                       on run module system
```

You are next ready to burn the run module into EPROM. This is done the same as the example on page 12-25 in the PDOS Reference manual.

In conclusion, run module generation is simpler with rev. 3.0b of the 9900 Pascal compiler. It is recommended that you first try a few simple programs before you attempt to EPROM your entire application.

FORCE COMPUTERS GmbH
Daimlerstraße 9
D-8012 Ottobrunn/München
Telefon 089/60091-0
Telefax (089) 6097793
Telex 524190 forc-d

Hypo-Bank München
BLZ 700 200 01
Konto-Nr. für DM: 3180238300
Konto-Nr. für US$: 9010179839

Registergericht München · HRB 67 465
Geschäftsführer Sven A. Behrendt

April 6, 1987/-pz

Dear PDOS User,

We are pleased to enclose the most recent issue of Eyring's

"PDOS Tips and Technical Notes"

At the same time we would like you to keep in mind that the bugs and limitations described in older notes have become invalid with new revisions of PDOS and compilers.

Yours sincerely,

FORCE COMPUTERS GMBH

Marketing Department

PDOS® Realtime Operating System

Tips & Technical Notes

Vol. 2 No. 2
March 2, 1987

INTRODUCTION

PRODUCT STATUS

1. Run Module Development. PDOS run module generation is the selective linking of user programs and the PDOS system modules to form a stand alone EPROMable application. A new RUNGEN utility available from Eyring simplifies this process. A descriptor file which describes the application is completed by using a template file as an example. The RUNGEN utility uses this descriptor file to produce a PDOS procedure file which generates the runable /EPROMable code. Run module applications can be programmed in Assembly, BASIC, C, PASCAL, or FORTRAN. This new run module development system should be used only with 3.2 PDOS and the latest versions of the various programming languages.

2. Current Product Versions. PDOS for the 68000 is currently at revision level 3.2a and supports the following language versions:

   FORTRAN 2.2d
   Pascal 3.0a
   C 5.0e
   BASIC 3.2a

PIG NEWS

The PDOS Interest Group (PIG) dial-in line at Eyring has been mostly available for the last few months. If you call (801) 375-2593 before 8:00 AM or after 5:00 PM (Mountain Time) you are connected to our PDOS modem. (You may also call (801) 375-2434 during our business hours, but you must have a voice connection on your phone and ask the operator for extension 264). We have had a very few calls, but we have also had trouble keeping the system up and going. Perhaps the reason that there wasn't much traffic is that the hardware was down. If you try to get in and can't, please give us a call.

The dial-in line answers at 2400/1200/300 baud, eight bits, no parity. A limited number of commands are allowed (SY, LS, SF, KERMIT, MF) for hacker control. The system presently has the current PIG disks 0 and 1 on-line (including the source to KERMIT) and a number of contributions that will make up PIG disk 2. We won't release PIG disk 2 until we have a full disk, (it's only a little over half full) but if you are interested, it currently has the following:

(PIG News cont.)

- A list of the attendees at the last user's group conference.
- A directory listing program that sorts by day of creation or update, extension, or name.
- A program that divides an EPROM image into upper/lower bytes for burning into a pair of ROMs.
- A program to control a Curtis EPROM burner.
- A program to show shared memory access under FORTRAN. (See Hints number 3).
- A set of login/logout utilities.
- A "Programmers Environment" shell that simplifies the edit/compile/debug cycle.
- A graphics game for the VME-1Ø.
- An exception handler that lets you specify an event to be automatically set every time a specific interrupt occurs.
- Another banner generator program with a gothic font.
- A program to split large files into smaller ones.
- An example of multiple programs periodically re-scheduling on the same timer event.
- A set of macros to aid in conversion of programs from 99ØØ assembly to 68ØØØ.

As usual with items in the public domain, this is all provided without any kind of support or maintenance, but if you want to perform your own improvements and resubmit them to PIG, we'd be happy to accept. Or, send in your own (non-proprietary) works of genius! You can send them over the dial-in line or on a disk to Brian Cooper at Eyring.

A new program is available on the 99ØØ PIG disk -- a programming tool to facilitate modular programming development in BASIC. Credit goes to Art Vreeland of Walker-Williams for development and to Millar Brainard for enhancements to the program BASCOMP. This is a pre-processor that accepts 'pseudo-basic' source with extensive comments and no line numbers, and strips the comments and adds line numbers. Labels for GOTO and GOSUB statements are supplied through a special macro-substitution facility. Code can be developed in multiple modules and then combined just before run time without worry about conflicting use of line numbers. It looks like a nice package for those doing development in BASIC.

Another interesting program from the 99ØØ group is BIRDS:SRC -- a geosynchronous satellite locator, courtesy of Mike Galvin of the Potomac Spaghetti Group. It used to be that only spies and astronomers watched satellites, but now, with satellite TV there is a larger interest. Anyone interested in converting either of these to 68ØØØ BASIC and checking them out should contact Brian Cooper at Eyring.

WARNINGS AND CAUTIONS

1. DOCUMENTATION - XSMP and XGMP were implemented differently than noted in the 3.1 documentation.

   DØ.B in the XSMP primitive is not a task number, but is a message slot number(Ø-15) into which the address of the message is placed. The task getting the message pointer uses the same message slot number to access the message. When the message is sent, the event number corresponding to the slot number+64 will be set. If no message is available when accessed with XGMP or if a message is already in the slot when XSMP is executed, the error message 83 will be in DØ and the status will be returned NE. Otherwise the status is returned EQ. The message is pointed to by (A1). Users should be aware of the potential conflict with the use of events 64 thru 79 in their application programs. Replace the pages of your manual with the attached corrected sheets. (The 3.2 documentation is correct).

2. DOCUMENTATION - The last two arguments in xwfp primitive as described on pages 3-11Ø and 3-111 of the PDOS C Reference Manual version 5.Ø were swapped. The parameter list should read as follows:

   ```
   int xwfp(eofsec,create,update,attr,filename);
   ```

   Replace the pages of your manual with those provided at the end of this document.

3. CAUTION - Some users have developed programs which have generated assembler error 324 (parameter out of range). This error results from long programs which generate return branches to locations beyond the range of the BRA.L addressing. To circumvent this error, write your code in smaller modules, or include GOTO statements which jump to locations for RETURN statements which will be within the addressing range.

4. NOTE - There is a change in the way the new assembler distributed with 5.Ø C and 3.2 PDOS handles register indirect with index instructions. In previous versions of the assembler, the displacement value could be left off if zero displacement was desired. The zero displacement value must now be included.

   ```
   MOVEA.L Ø(A2,D1.W),A2
           ^
   ```

(Warnings and Cautions cont.)

5. NOTE - The version of MTRANS distributed with 3.2 PDOS will not work properly under 3.0 PDOS. This is because the method used to set file parameters is different. The new version makes use of the new 3.2 PDOS primitive XWFP (write file parameters) for status duplication.

6. CAUTION - Registers A5 and A6 are corrupted when accessing the transcendental functions under PDOS Pascal. This should only have an effect on your programming if you are using static variables. We do not recommend using static variables under PDOS Pascal as there can be problems with this mode of variable access. Registers A5 and A6 will be saved in the next release of PDOS Pascal.

7. CAUTION - When a background task is writing information to the screen of another task, it is important that the writing task read the cursor position of the receiving task before writing to the screen. The task should restore the original cursor position to prevent disruption of the screen. Additionally, the task should lock itself or raise its priority to assure that the other task won't be sent. It is necessary for the background task to specify the port number when using the XRCP (read cursor position) primitive. If a zero is used, then the task will use the port number in the PRT field of the TCB. For a background task with no input port, this value will be zero, and the return values are unpredictable. A method for doing this in C is illustrated in the following task which writes the time to a fixed position on another task's screen:

```
#include <TCB:H>               /* define TCB structure */
char *xrtm();                  /* function returns string */
main()
{
  int port = _tcbptr->_u1p;    /* get output port number */
  asm("xcls");                 /* clear screen */
  while (1){                   /* loop forever */
     int row,col;              /* local variables */
     xsui(113);                /* wait on one second counter */
     xlkt;                     /* lock task */
     xrcp(port,&row,&col);     /* read receiving tasks parameters */
     xpsc(1,70);               /* move to screen position */
     xplc(xrtm());             /* print time */
     xpsc(row,col);            /* reposition to original location */
     xult;                     /* unlock task */
  }
}
```

FIXES, PATCHES AND WORKAROUNDS

1. PATCH - QLINK versions distributed with PDOS version 3.1 and C version 5.Ø had a problem which would give illegal object tag errors. This problem can be corrected with the following patch:

Use the patch which matches your QLINK distribution. Make a backup copy of your current QLINK.

3.1 QLINK (23-May-86):

```
>LO QLINK                   ;Load QLINK into memory
>PB                         ;Enter debugger
N1                          ;Byte mode
+93B    AE AD               ;Enter address and change code
+9CB    AE AD
+9F5    AE AD
+1ECF   6E 6D
+1ED3   2E 2D
+1ED9   2E 2D
+1EED   EE ED
+1EFA   3D 3B
+1FØ6   3D 3B
+1FØA   3D 3B
+1FØE   2D 2B
Q                           ;Exit debugger
>SV QLINK                   ;Save patched QLINK
>SA QLINK,SY                ;Make it an SY file
```

C 5.Ø QLINK (17-Oct-86):

```
>LO QLINK                   ;Load QLINK into memory
>PB                         ;Enter debugger
N1                          ;Byte mode
+AØF    AE AD               ;Enter address and change code
+AAD    AE AD
+B15    AE AD
+2627   6E 6D
+262B   2E 2D
+2631   2E 2D
+2645   EE ED
+2652   3D 3B
+265E   3D 3B
+2662   3D 3B
+2666   2D 2B
Q                           ;Exit debugger
>SV QLINK                   ;Save patched QLINK
>SA QLINK,SY                ;Make it an SY file
```

(Fixes, Patches, and Workarounds cont.)

2. FIX - The XGUM example in TESTXLIB:C assigns the variable freesize to be 2 blocks smaller than actually found. As a result, when recovering this memory using XFUM, all the memory is not recovered. To correct this problem, freesize should be increased by two. In TESTXLIB:C under the subroutine int doxgum() change the following:

```
{
   char *dummy;
   int i;
   long j;
   .
   .
   freesize = i + 2;      /* add 2 to freesize */
   freeptr = j;
   return Ø;
}
```

3. NOTE - The date format under PDOS 3.2 has changed from the mm/dd/yy format to the dd-mon-yr format. To read this new format, a new PDOS primitive was created. As a result, the reserved word $DATE which returns the assembly date is handled differently in the new MASM assembler. C users have received a copy of the latest assembler which uses the new format. If files are assembled using the new assembler, the reserved word $DATE will return a hex number under 3.Ø PDOS. Until you update to the 3.2 PDOS, use the older version of the assembler when the reserved word $DATE is used.

4. PATCH - On occasion, the square root function in PDOS Pascal would return incorrect values. This was due to the use of an LSR.B rather than the ASR.B. This error can be patched at the object level in the file PLIBF:LIB using MEDIT. Change both occurrences of "E2ØB" to "E2Ø3" and save the file. This problem will be corrected in the next release of PDOS Pascal.

APPLICATIONS AND HINTS

1. Alternate EPROM/RAM Disk Driver

The following code illustrates a method for implementing a disk driver which will access a disk on an EPROM or in user RAM. This driver can be used in addition to the RAM disk driver provided with PDOS. MOVE is a routine which transfers the data to and from disk and the PDOS buffers.

A file "DISKIMG" is created by saving the image of a RAM disk to a disk file.

```
>FM -10          ;free up 10 pages of memory for a 40 sector disk
  addr=00DF000
>RD -8,40,$DF000
```

Transfer the runable programs and any necessary libraries into the RAM disk area. Save the disk image to a disk file.

```
>SV #DISKIMG,$DF000,$E1800
```

This disk image may be converted to S-Records and loaded into a separate EPROM at a known address. If the user sets the file attribute to SY or converts the file to OB format with MSYOB, it may be loaded as part of the QLINK input.

```
>MSYOB
68K PDOS OB File Maker Utility 05-Dec-86
Source File = DISKIMG
Destination File = #DISKIMG:OBJ
```

The variable definitions to establish the size and location of the EPROM disk must be included as part of the QLINK portion of xxDOS:GEN which should be modified as follows:

```
 .
IN xxBIOSW:OBJ
DEFINE
W$EPDADR,Q$H0
DEFINE W$EPDN,<disk #>
DEFINE W$EPDSZ,<# sectors>
IN DISKIMG:OBJ
IN MSYRAM:OBJ
 .
```

An alternate method is to define W$EPADR at an address which is known when the memory is made available using the >FM command. The address of an EPROM disk image could also be used. Access to the disk is made available by using the >SY command.

(Applications and Hints cont.)

```
>FM 40
Address=A8000
>LO DISKIMG,$A8000
>SY <disk #>
```

Modify the xxBIOSW:SR file to include code similar to the following example. The code with minor modifications can read and write to a disk image loaded into user RAM. The disk is accessed like a normal disk except that it is not writeable if it is on EPROM. The code modifications are at the BIOSW entry points W$XWSE and W$XRSE.

```
**VARIABLE DEFINITIONS TO BE DECLARED DURING QLINK**

        XREF    W$EPDN    ;EPROM/RAM DISK NUMBER
        XREF    W$EPDSZ   ;EPROM/RAM DISK SIZE
        XREF    W$EPDADR ;START ADDRESS OF EPROM/RAM DISK

W$ENDADR        EQU     W$EPDADR+W$EPDSZ*256  ;END ADDRESS OF EPROM/RAM DISK


**READ/WRITE HANDLER - EPROM DISK READ ONLY**

**************************************************
*       WRITE SECTOR
*
*W$XWSE MOVE.W  #$0A30,D2         ;ORIGINAL CODE F1BIOSW:SR
*       BRA.S   COMMON
*
W$XWSE  MOVEQ.L #1,D2             ;SET EPROM DISK TO WRITE MODE
        CMPI.W  #EPDN,D0          ;IS IT EPROM/RAM DISK ?
          BEQ.S EPDRV             ;YES - GO PROCESS
        MOVE.W  #$0A30,D2         ;NO - GET SASI|WFC WRITE COMMANDS
        BRA.S   COMMON            ;PROCESS NORMALLY

EPDRV   MULU    #256,D1           ;CALCULATE BYTE OFFSET
        MOVEA.L #EPDADR,A1        ;GET EPROM DISK ADDRESS
        ADDA.L  D1,A1             ;GET DISK FINAL ADDRESS
        CMPA.L  #ENDADR,A1        ;IS IT TOO LARGE ?
         BGE.S  RDERR             ;YES - SET ERROR AND RETURN
        CMPI.L  #0,D2             ;READ FROM EPROM DISK ?
          BEQ.S @001              ;YES
*
**USE THIS CODE FOR EPROM DISK WRITE PROTECT
*
        MOVE.L  #103,D0           ;NO - SET WRITE PROTECT ERROR
        RTS                       ;RETURN WITH ERROR
*
```

(Applications and Hints cont.)

```
**USE THE FOLLOWING FOR RAM DISK WRITE ENABLED
*
*       EXG     A1,A2           ;NO - SWAP DIRECTION FOR WRITE
*
@001    BSR.L   MOVE            ;GO MOVE SECTOR DATA
        MOVE.L  #0,D0           ;SET STATUS RETURN .EQ.
        RTS                     ;RETURN
*
RDERR   MOVE.L  #101,D0         ;SET READ OVERFLOW ERROR
        RTS

*************************************************
*       READ SECTOR
*
*W$XRSE MOVE.W  #$0820,D2       ;OLD F1BIOSW:SR CODE
*
W$XRSE  CLR.L   D2              ;SET TO READ MODE
        CMPI.W  #EPDN,D0        ;IS IT EPROM/RAM DISK ?
          BEQ.S EPDRV           ;YES - GO PROCESS
        MOVE.W  #$0820,D2       ;NO - GET SASI|WFC READ COMMANDS
*
*       COMMON READ/WRITE FROM F1BIOSW:SR
*
COMMON  CLR.W   -(A7)           ;PUSH .NE.
        TST.W   P$SASF          ;EITHER CONTROLLER IN?
          BEQ   ERR100          ;N
        .
        .
        .

**DATA MOVER ROUTINE**

*************************************************
*       MOVE 256 BYTES OF DATA TO/FROM BUFFER
*
*         IN:   A1 = SOURCE
*               A2 = DESTINATION
*
MOVE    MOVEQ.L #256/4-1,D3     ;GET COUNT
*
MOVE2   MOVE.L  (A1)+,(A2)+     ;MOVE IT
         DBF    D3,MOVE2
        RTS
```

2. HINT - MINST reports the number of pages installed when new memory is installed. The pages represent the number of 2K byte blocks of memory added. The PDOS memory bit map uses one bit for each 2K bytes of memory. You must multiply the number of pages added by 2 if you are trying to determine whether you have installed all of the available memory.

(Applications and Hints cont.)

3. HINT - To allow sharing of GLOBAL variables across chained programs or between BASIC tasks, a special flag byte is available in PDOS BASIC. The clear/remark flag, in SYS[33], is a dual function flag: 1) strip REMarks from programs; and 2) clear variables as they are dimensioned. SYS[33] is set to zero when BASIC is entered or with the NEW command, and it is left unaltered by the CLEAR and RUN commands. The first function is performed at LOAD time. If SYS[33] is greater than zero (1 to 127), then as a file is LOADed, all remark strings are dropped from both REM statements and trailing remarks (!). This can be useful in creating smaller run module binary (BX) images from well commented ASCII (EX) programs, for example:

```
NEW                     Clear old programs, reset SYS[33]
*READY
SYS[33]=1               Set for REM strip
LOAD "WITH:REM"         Load 'EX' file, stripping REMs
*READY
SAVEB "WITHOUT:REM"     Save as 'BX' file with REMs stripped
```

The REMark stripping is only done when a file is LOADed or lines are entered from the terminal during editing, and therefore it cannot be used for stripping REMarks from binary "BX" files. If SYS[33] <- Ø (from -128 to Ø) at LOAD time, then remarks are preserved normally.

The second function of the clear/remark flag SYS[33] is performed as the program is running, at variable allocation time. When a variable, either simple or dimensioned, is first encountered in a program or GLOBAL statement, it is normally cleared to all zeroes. This complicates sharing GLOBAL variables between BASIC tasks. If SYS[33] >- Ø (from Ø to 127), then variables are zeroed as they are allocated. However if SYS[33] < Ø (-127 to -1) then the variables are NOT zeroed at allocation or GLOBAL time. As a result, these values may be passed from program to program, or from task to task. It is necessary however, that the sharing programs use the same order of GLOBAL or DIM variable definition to assure that their storage allocation is the same.

(Applications and Hints cont.)

```
PRGM1:  100  DIM CM(70)
        110  MAIL(0)=ADR CM(0)
        120  GLOBAL MAIL(0),A,B(10),C(10,4),VEL
        130  A=20: B(3)=30: C(5,1)=40: VEL=50
        140  PRINT A,B(3),C(5,1),VEL
        150  CLEAR ! Show nothing up this sleeve
        160  RUN "PRGM2"
PRGM2:  120  SYS[33]=-1  ! Preserve values the first time
        130  GLOBAL MAIL(0),A,B(10),C(10,4),VEL
        140  PRINT A,B(3),C(5,1),VEL
        150  SYS[33]=0  ! Destroy values this time
        160  GLOBAL MAIL(0),A,B(10),C(10,4),VEL
        170  PRINT A,B(3),C(5,1),VEL
        180  BYE
RESULTS:
>PRGM1
 20             30              40              50
 20             30              40              50
 0              0               0               0
>_
```

4. APPLICATION - In Vol. 1 No. 3 of "PDOS Tips and Technical Notes," a method was described to change the TPS (tics per second) to 1000. It was assumed that the clock would interrupt on the 10,000 of seconds which was not the case. As a result, this patch did not work. The following application, which was implemented in the FORCE CPU-1 BIOS, illustrates a method for selecting TPS of 100, 125, 200, 250, or 500. Unfortunately, 1000 TPS is not possible on the RTC.

```
        ...
        IFUDF   RTCF    :RTCF   EQU 0   ;(0) PI/T == TIMER
*                                        (1) RTC === TIMER
*
RTCC    EQU     1000/TPS        ;GET A CONSTANT (=ms INCREMENTER)
        IFNE    RTCF&(1000<>TPS*RTCC)!(RTCC<2)!(RTCC>10)
        FAIL    Bad TPS value for RTC clock.  Use 100,125,200,250, or 500
        ENDC    ;RTCF
*
        OPT     ARS,CRE
        ...
```

(Applications and Hints cont.)

```
        ...
BINT6   TST.B    RTC+ISR          ;RESET RTC INT IN EITHER CASE
        TST.B    PIRV+PI_T        ;WAS RTC INT FOR TIMER? ($FF=PI/T,$00=RTC)
        BEQ.S    @010             ;RTC, SET TPS
        RTE                       ;NOT RTC, JUST RETURN
*
@010    MOVE.L   D0,-(A7)         ;SAVE REG
        MOVEQ.L  #0,D0            ;CLEAR D0
        MOVE.B   RAM+RTC,D0       ;GET COUNTER BYTE
        ADDI.W   #RTCC<<4,D0      ;UP .001 COUNTER COMPARE
        CMPI.W   #10<<4,D0        ;OVERFLOW ?
          BLO.S  @020             ;N
        SUBI.W   #10<<4,D0        ;COMPARE NEXT AT MODULO 10
*
@020    MOVE.B   D0,RAM+RTC       ;RESTORE VALUE
        MOVE.L   (A7)+,D0         ;RESTORE D0
        BRA.L    K1$CLKI          ;RTC: DROP TO KERNEL CLOCK ROUTINE
*
        ...
```

The TPS parameter must be initialized to the new value and the system regenerated using the FxDOS:GEN procedure file.

5. APPLICATION - A PDOS user, Ron Stear of PPG, recently provided an application example which permits the using of global variables in Absoft FORTRAN. His application provides a means for setting up the global area and passing this information to other tasks in the MAIL array. The tasks accessing this area need only know the offset of the variable blocks. Ron illustrates in an example program the means for writing to and reading these variables from other tasks. The program BASE is created as a task which obtains a block of free memory. The task determines its starting location, passes this to the mail array and kills itself. The example is written to allow recovery of the memory, i.e. it is returned to the free memory pool. In some applications you may want to kill the task using a negative task number to prevent deallocation of the memory.

   This program is available on PIG disk number 2 (currently available through the call-in modem -- see PIG News).

   In Ron's example program, he executes the XCTB primitive to create the BASE task. This would be done in only one task to set up the global memory area. The program appears following:

```
        PROGRAM BASE
        INTEGER TASKNO,XRTS,ERROR,START,END,XKTB,XBUG
C SYSTEM CHARACTER ARRAYS
        CHARACTER*40 STRINGS(10)
C SYSTEM INTEGER ARRAYS
        INTEGER INUMBERS(10)
C SYSTEM REAL ARRAYS
        REAL*8 RNUMBERS(10)
C
        INTEGER MAILPTR
        INTEGER ENDTCB,UPPERMEM,LASTLOAD,SYRAM,TCB
C
C IF YOU WILL RUN F77,A ON THIS 'BASE' PROGRAM, IT WILL PRODUCE THE
C ASSEMBLY LISTING OF THE FORTRAN CODE.
C THIS LISTING MAY BE EXAMINED TO FIND THE 'OFFSET' TO EACH OF THE
C ABOVE DEFINED DATABASE AREAS WHICH MUST THEN BE IDENTIFIED IN EACH
C OF THE THREE SUBROUTINES.
C
C THIS IS NOT REALLY NECESSARY SINCE THE ALLOCATED MEMORY WAS OBTAINED
C WHEN THE 'BASE' TASK WAS CREATED AND IT WILL NOT ACTUALLY OCCUPY
C ALL OF THE CREATED MEMORY SIZE.
C
C THE OFFSETS USED IN EACH SUBOUTINE MAY BE CALCULATED BY MULTIPLYING
C THE NUMBER OF ELEMENTS BY THE NUMBER OF 'STORAGE UNITS' OR WORDS
C FOR EACH DEFINED TYPE.
C
C THEREFORE, FOR THIS DEMONSTRATION PROGRAM:
C
C       THE OFFSET TO THE STRINGS IS    START                   (0)
C       THE OFFSET TO THE INTEGERS IS
C               NUMBER OF STRINGS*40    STRINGS+STRING SIZE     (400) 190H
C       THE OFFSET TO THE REALS IS
C               NUMBER OF INTEGERS*4    INTEGERS+INTEGER SIZE   (800) 320H
C
C THESE VARIABLES NEED ONLY BE SET AS PARAMETERS IN THE SUBROUTINES
C WHICH CALCULATE THE LOAD ADDRESS (LOADADDR).
C
        STRINGS(1)(1:1)='1'
        INUMBERS(1)=2222
        RNUMBERS(1)=3333
        CALL XGML(ENDTCB,UPPERMEM,LASTLOAD,SYRAM,TCB)
        MAILPTR=LONG(SYRAM+4)                   !ADDRESS OF MAIL
        LONG(MAILPTR)=ENDTCB                    !PUT ADDRESS IN MAIL
        ERROR=XKTB(XRTS(-1))
        STOP
        END
```

```
C PROGRAM DEMO
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C PROGRAM:      DEMO
C DATE:         11/3/86
C FUNCTION:     DEMONSTRATE 'COMMON' ACCESS
C PROGRAMMER:   RONALD B. STEAR
C               PPG INDUSTRIES, INC
C               PO BOX 1000
C               LAKE CHARLES, LOUISIANA 70602
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        INTEGER BASE            !ADDRESS OF DATABASE
        INTEGER MAILPTR         !POINTER TO MAIL
        INTEGER ENDTCB,UPPERMEM,LASTLOAD,SYRAM,TCB
C
        INTEGER XCTB            !CREATE TASK BLOCK FUNCTION
        INTEGER XSUI            !SUSPEND INTERRUPT FUNCTION
        CHARACTER*80 XGLM       !GET LINE IN MONITOR FUNCTION
C
        INTEGER INUMBER         !INTEGER NUMBER TO PLAY WITH
        REAL*8 RNUMBER          !REAL NUMBER TO PLAY WITH
        CHARACTER*40 STRING     !STRING TO PLAY WITH
C
        INTEGER ERROR,COUNT,TASKNO,ELEMENT
C
        INTEGER GET,PUT         !COMMANDS TO SAVE & RETRIEVE
        PARAMETER(GET=0, PUT=1)
C
C FIRST CREATE THE 'BASE' TASK TO RESERVE MEMORY FOR THE DATABASE
C
        ERROR=XCTB(TASKNO,100,64,0,'BASE',0,0)
C
C THEN WAIT A MOMENT FOR IT TO KILL ITSELF
C
        COUNT=1
        WHILE(COUNT.LT.10)
          ERROR=XSUI(112)
          COUNT=COUNT+1
        REPEAT
C
C FIND OUT WHERE THE DATABASE IS
C
        CALL XGML(ENDTCB,UPPERMEM,LASTLOAD,SYRAM,TCB)
        MAILPTR=LONG(SYRAM+4)
        BASE=LONG(MAILPTR)
C
```

```
C SINCE THE MEMORY IS SIMPLY ALLOCATED, IT MUST FIRST BE INITIALIZED
C SO THAT THE UPDATE SUBROUTINE WON'T PRINT GARBAGE ALL OVER THE SCREEN
C THIS IS NOT NECESSARY IF YOU ARE GOING TO INITIALIZE THE DATA BEFORE
C USING IT OR IF YOU LOAD THESE ARRAYS FROM A DISK FILE
C
        ELEMENT=1
        WHILE(ELEMENT.LE.10)
          COUNT=1
          WHILE(COUNT.LE.39)
            STRING(COUNT:COUNT)=' '
            COUNT=COUNT+1
          REPEAT
          STRING(40:40)=CHAR(0)
          CALL TEXT(BASE,PUT,ELEMENT,STRING)
          CALL IBASE(BASE,PUT,ELEMENT,0)
          CALL RBASE(BASE,PUT,ELEMENT,0)
          ELEMENT=ELEMENT+1
        REPEAT
C
C CLEAR OFF THE SCREEN AND BEGIN THE DEMONSTRATION
C
10      CALL XCLS
        TYPE *,' "BASE" PROGRAM HAS ALLOCATED MEMORY STARTING AT '
        WRITE (9,11) BASE,'H'
11      FORMAT (Z8,A)
        CALL XPSC(3,5)
        TYPE *,'1. INTEGER NUMBERS'
        CALL XPSC(4,5)
        TYPE *,'2. REAL NUMBERS'
        CALL XPSC(5,5)
        TYPE *,'3. 40 CHARACTER STRINGS'
        CALL XPSC(6,5)
        TYPE *,'4. QUIT'
C
15      CALL XPSC(7,5)
        TYPE *,   '                                 '
        TYPE *,   '                                                '
        CALL XPSC(7,5)
        CALL INUM('SELECT OPTION NUMBER (1-4) >',COUNT,ERROR)
        IF((COUNT.LT.1).OR.(COUNT.GT.4)) THEN
          TYPE *,CHAR(7)
          GOTO 10
        END IF
C
```

(Applications and Hints cont.)

```
        SELECT CASE COUNT
          CASE(1)
20          CALL XPSC(7,5)
            TYPE *,    '                                        '
            CALL XPSC(7,5)
            CALL INUM('ENTER ELEMENT NUMBER (1-10)>',ELEMENT,ERROR)
            IF((ELEMENT.LT.1).OR.(ELEMENT.GT.10)) THEN
              TYPE *,CHAR(7)
              GOTO 20
            END IF
            CALL XPSC(7,5)
            TYPE *,    '                                        '
            CALL XPSC(7,5)
            CALL INUM('ENTER INTEGER NUMBER       >',INUMBER,ERROR)
            CALL IBASE(BASE,PUT,ELEMENT,INUMBER)
          CASE(2)
30          CALL XPSC(7,5)
            TYPE *,    '                                        '
            CALL XPSC(7,5)
            CALL INUM('ENTER ELEMENT NUMBER (1-10)>',ELEMENT,ERROR)
            IF((ELEMENT.LT.1).OR.(ELEMENT.GT.10)) THEN
              TYPE *,CHAR(7)
              GOTO 30
            END IF
            CALL XPSC(7,5)
            TYPE *,    '                                        '
            CALL XPSC(7,5)
            CALL RNUM('ENTER REAL NUMBER          >',RNUMBER,ERROR)
            CALL RBASE(BASE,PUT,ELEMENT,RNUMBER)
          CASE(3)
40          CALL XPSC(7,5)
            TYPE *,    '                                        '
            CALL XPSC(7,5)
            CALL INUM('ENTER ELEMENT NUMBER (1-10)>',ELEMENT,ERROR)
            IF((ELEMENT.LT.1).OR.(ELEMENT.GT.10)) THEN
              TYPE *,CHAR(7)
              GOTO 40
            END IF
            CALL XPSC(7,5)
            TYPE *,    '                                                 '
            CALL XPSC(7,5)
            TYPE *,'ENTER STRING >'
            STRING=XGLM(ERROR)
            CALL TEXT(BASE,PUT,ELEMENT,STRING)
```

```
          CASE(4)
            CALL XCLS
            STOP
          CASE DEFAULT
            TYPE *,CHAR(7)
            GOTO 10
        END SELECT
        CALL UPDATE(BASE)               !UPDATE THE DEMO SCREEN
        GOTO 15
        END
C SUBROUTINE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C SUBROUTINE:   UPDATE
C DATE:         11/3/86
C FUNCTION:     UPDATE THE SCREEN TO SHOW ALL 10 ELEMENTS OF EACH
C               ACTIVE DATABASE SECTION
C PROGRAMMER:   RONALD B. STEAR
C               PPG INDUSTRIES, INC.
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE UPDATE(BASE)
C
        INTEGER BASE,COUNT,INUMBER
        REAL*8 RNUMBER
        CHARACTER*40 STRING
        INTEGER GET,PUT,C
        PARAMETER(GET=0, PUT=1)
C
        CALL CLEAR      !CLEAR DATA AREA OF SCREEN BEFORE UPDATE
        CALL XPSC(9,1)
        TYPE *,'INTEGERS       REALS            STRINGS'
        COUNT=1
        WHILE(COUNT.LE.10)
          CALL XPSC(COUNT+9,1)
          CALL IBASE(BASE,GET,COUNT,INUMBER)
          TYPE *,INUMBER,'    '
          CALL XPSC(COUNT+9,16)
          CALL RBASE(BASE,GET,COUNT,RNUMBER)
          TYPE *,RNUMBER
          CALL XPSC(COUNT+9,33)
          CALL TEXT(BASE,GET,COUNT,STRING)
          C=1
          WHILE((STRING(C:C).NE.CHAR(0)).AND.(C.LE.40))
            TYPE *,STRING(C:C)
            C=C+1
          REPEAT
          COUNT=COUNT+1
        REPEAT
        RETURN
        END
```

(Applications and Hints cont.)

```
C SUBROUTINE IBASE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C SUBROUTINE:   IBASE
C DATE:         11/3/86
C FUNCTION:     ACCESS TO INTEGER 'DATABASE'
C PROGRAMMER:   RONALD B. STEAR
C               PPG INDUSTRIES, INC.
C               PO BOX 1000
C               LAKE CHARLES, LOUISIANA  70602
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
        SUBROUTINE IBASE(BASE,GP,ELEMENT,INUMBER)
C
        INTEGER BASE,GP,ELEMENT,INUMBER
        INTEGER LOADADDR,OFFSET
        INTEGER GET,PUT
        PARAMETER(GET=0, PUT=1)
C
C THE PARAMETER BELOW IS THE CALCULATED OFFSET INTO MEMORY FROM THE
C BEGINNING OF THE 'BASE' TASK'S PROGRAM AREA TO THE INTEGER AREA AND
C MUST BE ESTABLISHED FOR EACH DATABASE CONFIGURATION
C
        INTEGER XINUM
        PARAMETER(XINUM=400)
C
C CALCULATE THE OFFSET INTO THE INTEGER MEMORY SEGMENT
C
        OFFSET=(ELEMENT-1)*2
        LOADADDR=BASE+OFFSET+XINUM
        IF(GP.EQ.GET) THEN
          INUMBER=WORD(LOADADDR)
        ELSE
          WORD(LOADADDR)=INUMBER
        END IF
        RETURN
        END
C SUBROUTINE RBASE
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C SUBROUTINE:   RBASE
C DATE:         11/3/86
C FUNCTION:     ACCESS TO 'REAL' DATABASE
C PROGRAMMER:   RONALD B. STEAR
C               PPG INDUSTRIES, INC.
C               PO BOX 1000
C               LAKE CHARLES, LOUISIANA  70602
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
CC
        SUBROUTINE RBASE(BASE,GP,ELEMENT,RNUMBER)
C
        INTEGER BASE,GP,ELEMENT,OFFSET,LOADADDR
        REAL*8 RNUMBER,XRNUM
        INTEGER GET,PUT
        PARAMETER(GET=0, PUT=1)
C
C THE PARAMETER BELOW IS THE CALCULATED OFFSET INTO MEMORY FROM THE
C BEGINNING OF THE 'BASE' TASK'S PROGRAM AREA TO THE 'REAL' AREA AND
C MUST BE ESTABLISHED FOR EACH DATABASE CONFIGURATION
C
        INTEGER REALOFFSET
        PARAMETER(REALOFFSET=800)
C
C REAL NUMBER TRANSFER INTEGER ARRAY
C
        INTEGER R(2)
C
C THE EQUIVALENCE ALLOWS STORAGE OF THE REAL NUMBERS WITH THE INTEGER
C FUNCTION 'LONG'
C
        EQUIVALENCE(R(1),XRNUM)
C
C CALCULATE THE OFFSET INTO THE REAL NUMBER MEMORY SEGMENT
C
        OFFSET=(ELEMENT-1)*8
        LOADADDR=BASE+OFFSET+REALOFFSET
        IF(GP.EQ.GET) THEN
          R(1)=LONG(LOADADDR)
          R(2)=LONG(LOADADDR+4)
          RNUMBER=XRNUM
        ELSE
          XRNUM=RNUMBER
          LONG(LOADADDR)=R(1)
          LONG(LOADADDR+4)=R(2)
        END IF
        RETURN
        END
C SUBROUTINE    TEXT
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C SUBROUTINE:   TEXT
C VERSION:      STRIDE 440
C DATE:         11/3/86
C FUNCTION:     CHARACTER DATA TRANSFER FROM DATABASE
C PROGRAMMER:   RONALD B. STEAR
C               PPG INDUSTRIES, INC.
C               PO BOX 1000
C               LAKE CHARLES, LOUISIANA  70602
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
```

```
C
        SUBROUTINE TEXT(BASE,GP,ELEMENT,STRING)
C
        INTEGER BASE,GP,ELEMENT
        CHARACTER*40 STRING,XSTRING
        INTEGER GET,PUT
        PARAMETER(GET=0, PUT=1)
C
C THE PARAMETER BELOW IS THE CALCULATED OFFSET INTO MEMORY FROM THE
C BEGINNING OF THE 'BASE' TASK'S PROGRAM AREA TO THE STRING AREA AND
C MUST BE ESTABLISHED FOR EACH DATABASE CONFIGURATION
C
        INTEGER XSTRN
        PARAMETER(XSTRN=0)
C
        INTEGER LOADADDR,OFFSET
C INTEGERS FOR 40 CHARACTER TRANSFERS
        INTEGER D1,D2,D3,D4,D5,D6,D7,D8,D9,D10
C EQUIVALENCES FOR 40 CHARACTER TRANSFERS
        EQUIVALENCE(XSTRING(1:4),D1)
        EQUIVALENCE(XSTRING(5:8),D2)
        EQUIVALENCE(XSTRING(9:12),D3)
        EQUIVALENCE(XSTRING(13:16),D4)
        EQUIVALENCE(XSTRING(17:20),D5)
        EQUIVALENCE(XSTRING(21:24),D6)
        EQUIVALENCE(XSTRING(25:28),D7)
        EQUIVALENCE(XSTRING(29:32),D8)
        EQUIVALENCE(XSTRING(33:36),D9)
        EQUIVALENCE(XSTRING(37:30),D10)
C
C CALCULATE THE OFFSET FOR THE STRING
C
        OFFSET=(ELEMENT-1)*40
        LOADADDR=BASE+OFFSET+XSTRN
C
        IF(GP.EQ.GET) THEN
          D1=LONG(LOADADDR)
          D2=LONG(LOADADDR+4)
          D3=LONG(LOADADDR+8)
          D4=LONG(LOADADDR+12)
          D5=LONG(LOADADDR+16)
          D6=LONG(LOADADDR+20)
          D7=LONG(LOADADDR+24)
          D8=LONG(LOADADDR+28)
          D9=LONG(LOADADDR+32)
          D10=LONG(LOADADDR+36)
          STRING(1:40)=XSTRING(1:40)
          RETURN
```

```
        ELSE
          XSTRING(1:40)=STRING(1:40)
          LONG(LOADADDR)=D1
          LONG(LOADADDR+4)=D2
          LONG(LOADADDR+8)=D3
          LONG(LOADADDR+12)=D4
          LONG(LOADADDR+16)=D5
          LONG(LOADADDR+20)=D6
          LONG(LOADADDR+24)=D7
          LONG(LOADADDR+28)=D8
          LONG(LOADADDR+32)=D9
          LONG(LOADADDR+36)=D10
          RETURN
        END IF
        END
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    SUBROUTINE:        INUM
C    VERSION:           STRIDE
C    DATE:              11/20/85
C    AUTHOR:            RONALD B. STEAR
C                       PPG INDUSTRIES, INC.
C                       PO BOX 1000
C                       LAKE CHARLES, LOUISIANA  70602
C    FUNCTION: INPUT AN INTEGER VALUE FROM A PROMPT LINE
C    CALL:
C                       CALL INUM(PROMPT,VAR,ERROR)
C                       WHERE:
C                       PROMPT IS A CHARACTER MESSAGE
C                       VAR IS AN INTEGER VARIABLE
C    RETURNS:
C                       INTEGER VARIABLE IN 'VAR'
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
      SUBROUTINE INUM(PROMPT,VAR,ERROR)
      IMPLICIT INTEGER (A-Z)
      CHARACTER PROMPT *(*)
      CHARACTER NULL,CR,LF,A,XULL
      PARAMETER (NULL=0,CR=13,LF=10,XULL=128)
      CHARACTER *132 LINE
      TYPE *,PROMPT
      ERROR=XGLB(LINE)
      IF(ERROR.EQ.0) RETURN
      CALL XCDB(LINE,VAR)
      RETURN
      END
```

```
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C    SUBROUTINE:         RNUM
C    VERSION:            STRIDE
C    DATE:               12/3/85
C    AUTHOR:             RONALD B. STEAR
C                        PPG INDUSTRIES, INC.
C                        PO BOX 1000
C                        LAKE CHARLES, LOUISIANA  70602
C    FUNCTION:           ACCEPT REAL NUMBER FROM KEYBOARD
C
CCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCCC
C
C
C
        SUBROUTINE RNUM(PROMPT,VAR,ERROR)
        IMPLICIT INTEGER(A-Z)
        REAL*8 VAR,FRACT,DCOUNT,DEC
        CHARACTER PROMPT *(*)
        CHARACTER CR,LF,CHAR,MINUS,POINT,BELL,ZERO,NINE,TERM
        PARAMETER(CR=13, LF=10, MINUS='-', POINT='.',BELL=7)
        PARAMETER(ZERO='0', NINE='9', TERM='|')
        CHARACTER WHOLE(10)
        CHARACTER DECIMAL(10)
        CHARACTER LINE(20)
C
        DO 20 J=1,10
        WHOLE(J)=0
        DECIMAL(J)=0
        LINE(J)=TERM
        LINE(J*2)=TERM
20      CONTINUE
        TYPE *,PROMPT
        ERROR=XGLB(LINE)
        IF(ERROR.EQ.0) RETURN
        PFLAG=0
        CCOUNT=1
        WCOUNT=0
        DCOUNT=0
        SIGN=1
C
10      CHAR=LINE(CCOUNT)
        IF(LINE(CCOUNT+1).EQ.TERM) GO TO 1000
        IF(CCOUNT.NE.1) GO TO 2000
        IF(CHAR.NE.MINUS) GO TO 2000
        SIGN=(-1)
4000    CCOUNT=CCOUNT+1
        GO TO 10
```

```
2000    IF(PFLAG.EQ.0) GO TO 3000
        IF((CHAR.LT.ZERO).OR.(CHAR.GT.NINE)) THEN
          TYPE *,BELL
          VAR=-.9999
          RETURN
        END IF
        DCOUNT=DCOUNT+1
        DECIMAL(DCOUNT)=CHAR
        GO TO 4000
C
3000    IF(CHAR.NE.POINT) GO TO 5000
        PFLAG=1
        GO TO 4000
C
5000    IF((CHAR.LT.ZERO).OR.(CHAR.GT.NINE)) THEN
          TYPE *,BELL
          VAR=0
          RETURN
        END IF
        WCOUNT=WCOUNT+1
        WHOLE(WCOUNT)=CHAR
        GO TO 4000
C
1000    W=0
        IF(WCOUNT.NE.0) CALL XCDB(WHOLE,W)
        D=0
        CALL XCDB(DECIMAL,D)
        DEC=FLOAT(D)
        FRACT=DEC*(10**(-DCOUNT))
        VAR=(FLOAT(W)+FRACT)*SIGN
        RETURN
        END

C****************************************
        SUBROUTINE CLEAR
C****************************************
C**CLEAR SCREEN FOR TV-925

        CHARACTER  CLR
        PARAMETER (CLR=27+256*89)       !<ESC>Y
C
        TYPE *,CLR
C**GENERAL PURPOSE CLEAR SCREEN
```

(Applications and Hints cont.)

```
C       INTEGER CNT

C       CNT = 10
C       WHILE (CNT .LE. 20)
C         CALL XPSC(CNT,1)
C         TYPE *,'                                        '
C         TYPE *,'                                        '
C         CNT=CNT+1
C       REPEAT
        RETURN
        END
```

4.3.42 XGMP - GET MESSAGE POINTER

```
     Mnemonic:     XGMP
        Value:     $A004
       Module:     MPDOSK1
       Format:     XGMP
                     <status return>

    Registers: In  D0.L = Message slot number (0..15)
               Out D0.L = Source task # (-1 = no message)
                     SR = EQ....Message (Event[64+Message slot #]=0)
                          NE....No message
                   D0.L = Error number 83 if no message
                   (A1) = Message
```

The GET MESSAGE POINTER primitive looks for a task message pointer. If no message is ready, then data register D0 returns with a minus one (-1) and status is set to 'Not Equal'.

If a message is waiting, then data register D0 returns with the source task number, address register A1 returns with the message pointer, event (64 + message slot #) is set to zero indicating message received, and status is returned equal.

See also:

4.3.44 XGTM - GET TASK MESSAGE
4.3.48 XKTM - KILL TASK MESSAGE
4.3.96 XSMP - SEND MESSAGE POINTER
4.3.99 XSTM - SEND TASK MESSAGE

Possible Errors:

83 = Message slot empty

4.3.43 XGNP - GET NEXT PARAMETER

```
      Mnemonic:     XGNP
         Value:     $A05A
        Module:     MPDOSM
        Format:     XGNP
                      <status return>

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

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

The GET NEXT PARAMETER primitive parses the monitor buffer for the next command parameter. The routine does this by maintaining a current pointer into the command line buffer (CLB$) and a parameter delimiter (CMD$).

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

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

A 'LO' status is returned if the last parameter delimiter is a null or period. XGNP does not parse past a period. In this case, address register A1 is returned pointing to a null string.

```
x>LS.LS
```

An 'EQ' status is returned if the last parameter delimiter is a comma and no parameter follows. Address register A1 is returned pointing to a null string.

```
x>MASM SOURCE,,,ERR
```

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

Possible Errors: None

4.3.94 XSEV - SET EVENT FLAG

```
   Mnemonic:    XSEV
      Value:    $A046
     Module:    MPDOSK1
     Format:    XSEV
                  <status return>

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

Note: Event 128 is local to each task.

If D1.B is positive, then the event is set.
If D1.B is negative, then the event is reset.

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

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

The SET EVENT FLAG primitive sets or resets an event flag bit. The event number is specified in data register D1.B and is modulo 128. If the content of register D1.B is positive, then the event bit is set to 1. Otherwise, the bit is reset to 0. Event 128 can only be set. (It is cleared by the task scheduler.)

The status of the event bit prior to changing the event is returned in the status register. If the event was 0, then the 'EQ' status is returned. A context switch DOES NOT occur with this call making it useful for interrupt routines outside the PDOS system.

4 types of event flags:

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

Events are summarized as follows:

1-63 =	Software events
64-80 =	Software resetting events
81-95 =	Output port events
96-111 =	Input port events
112 =	1/5 second event
113 =	1 second event
114 =	10 second event
115 =	20 second event
116 =	TTA active
117 =	LPT active
118 =	
119 =	
120 =	Level 2 lock
121 =	Level 3 lock
122 =	Batch event
123 =	Spooler event
124 =	
125 =	
126 =	Error message disable
127 =	System utility
128 =	Local

See also:

4.3.20 XDEV - DELAY SET/RESET EVENT
4.3.95 XSEV - SET EVENT FLAG
4.3.101 XSUI - SUSPEND UNTIL INTERRUPT
4.3.106 XTEF - TEST EVENT FLAG

Possible Errors: None

4.3.95 XSMP - SEND MESSAGE POINTER

```
Mnemonic:      XSMP
   Value:      $A002
  Module:      MPDOSK1
  Format:      XSMP
                 <status return>

Registers: In  D0.B = Message slot number (0..15)
               (A1) = Message
           Out   SR = EQ....Message sent (Event[64+slot #]=1)
                      NE....No message sent
```

The SEND MESSAGE POINTER primitive sends a 32-bit message to the message slot specified by data register D0.B. Address register A1 contains the message.

If there is still a message pending, then the primitive immediately returns with status set 'Not Equal' and D0.L equal to 83. Otherwise, the message is taken by PDOS event (64 + message slot number) is set to one indicating a message is ready, and status is returned 'Equal'.

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

See also:

4.3.42 XGMP - GET MESSAGE POINTER
4.3.44 XGTM - GET TASK MESSAGE
4.3.48 XKTM - KILL TASK MESSAGE
4.3.99 XSTM - SEND TASK MESSAGE

Possible Errors:

83 = Message buffer pending

XWFA

Write File Attributes

Format:

```
int xwfa(filename,attributes);
char *filename,*attributes;
```

Description:

XWFA sets the file attributes on a file. The ASCII string of file attributes is assigned to the file. Any errors are returned; Ø is returned if there are no errors.

```
AC - Procedure file
BN - Binary file
OB - Object file
SY - Memory Image of machine code
BX - BASIC token file
EX - BASIC ASCII file
TX - Text file
DR - System I/O driver

*  - Delete protect
** - Delete/write protect
```

```
int err;
err = xwfa("MYFILE","BN**");  /* make file binary and protected */
```

Notes:

XCFA, XRFA, and XWFA do not use the same format.

See Also:

XCFA - Close file w/attribute
XRFA - Read file attributes

XWFP

Write File Parameters

Format:

```
int xwfp(eofsec,create,update,attr,filename);
long eofsec;                 /* sector / byte */
long create,update,attr;     /* time / date */
char *filename;
```

Description:

XWFP is an operating system internal call used by the TF monitor command to assign a copy of a file the same creation/update time and date as the original of the file. It could also be used to modify the end of file mark on a file.

The first three parameters are all pairs of data. 'eofsec' is a long word with the end of file sector in the upper word and the end of file byte in the lower word. 'create' is a long word with the creation time in the upper word and the creation date in the lower word. 'update' is in the same format as 'create'. 'filename' is a pointer to a string containing the file name. 'attr' is a long word, but only the second half is used. This word has the attributes in the upper byte and the delete-/write-protect flags in the lower byte. The "contiguous" flag and the "file altered bit" are not overwritten by this call. 'xwfp' returns zero or a PDOS error number.

```
union{
    long l;
    struct {
        int time;
        int date;
    };
} create,update;
union{
    long l;
    struct{
        int sector;
        int byte;
    };
} eof;
```

continued . . .

```
char line[80];
long attr;
getstr("enter file name",line);     /* get new values */
eof.sector = getnum("end of file sector");
eof.byte = getnum("end of file byte");
create.time = getnum("creation time");
create.date = getnum("creation date");
update.time = getnum("update time");
update.date = getnum("update date");
attr = getnum("attribute");
return(xwfp(eof.l,create.l,update.l,attr,line));     /* write it */
}
```

Notes:
This function has limited utility.

See Also:
XRFA - Read file attributes

XWLF

Write Line to File

Format:

```
int xwlf(filid,buffer);
int filid;
char *buffer;
```

Description:

XWLF writes a string to a file. It writes out until a null character is found. If necessary, a contiguous file is extended and converted to a non-contiguous file. Any errors are returned, and a return value of Ø means no errors.

```
  err = xwlf(filid,"hello, world!/n");
```

See Also:

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

XWBF - Write bytes to file

FORCE COMPUTERS®

March 7, 1988

PDOS REAL TIME OPERATING SYSTEM

VOLUME 1 MANUAL UPDATE

Please append the following chapter (**PDOS TIPS and TECHNICAL NOTES**) in your User's Manual:

Chapter 8: USER NOTE 1.

Thank you.

FORCE COMPUTERS GmbH

PDOS TIPS
and
TECHNICAL NOTES

Volume 3 No. 1
March 1, 1988

The information in this document has been carefully checked and is believed to be reliable. However, Eyring assumes no responsibility for inaccuracies. Furthermore, Eyring reserves the right to make changes to any products to improve reliability, function, or design and does not assume any liability arising out of the application or use of this document.

PDOS Tips and Technical Notes, Volume 3 Number 1

Printed in the United States of America
Product number 2553-1 (for PDOS 3.3)
March 1, 1988

Table of Contents

Product Status

Update

PDOS 3.3

The PDOS 3.3 update has been released. It includes significant improvements in the software as well as new documentation. You should have received notification about the update. Requests for updates are currently being filled.

Update

C Compiler

PDOS C has been updated to version 5.4 to run with PDOS 3.3. Some changes include new definitions of math.h, changes in the libraries and include files, and bug fixes.

Warnings and Cautions

Note

Proper year for battery and PDOS clocks

If you have not already updated your system for 1988, change your SY$STRT file to include this year in the MTIME utility parameter.

```
MTIME P,88
```

Note

TM and a modem

It doesn't make sense to have two programs reading from the same port at the same time. PDOS attempts to prevent this by not allowing you to create two tasks with the same input port. It is possible, however, to have one task running on a remote port and enter transparent mode from the monitor (the TM command) to talk to that port. When this happens, the task running on the port gets some of the characters while the monitor in transparent mode gets the rest. The result is confusion. Kill the task running on a remote port before you go into transparent mode on that port.

Note

MEDIT with 8-bit characters

MEDIT cannot handle 8-bit characters at present. Users who have tried to use MEDIT on 8-bit data have failed, partly because MEDIT uses the eighth bit internally for pointer information. If this is a problem for you, call customer support.

Note

XVEC returns pointer to jump table

In ROM-based interrupt vector systems, XVEC with A0=0 returns a pointer to the jump table in RAM rather than the address of the interrupt handler. This is proper behavior since it would require writing to ROM with possible errors if it were otherwise. XVEC with A0 containing the desired address or old address properly returns the pointer to the interrupt handler. To read the vector pointer, rewrite the interrupt handler address to the vector.

Caution

BASIC - powers of negative numbers

In BASIC, powers of negative numbers do not generate the proper result. In the latest release of PDOS, improper error messages may also result.

Caution

Assembly ENDM directive

The ENDM directive is case sensitive and must be included as capital letters. If the statement is not present or not found, PDOS error 56 is generated as an indication that a problem has occurred. This will be corrected in a future version of the assembler.

Caution

XDEF following OFFSET

The PDOS linker does not properly find an XDEF statement when it follows an OFFSET statement. The linker is unable to resolve the XDEF under this condition. It is necessary to declare XDEFs before any OFFSET statements are used. This will be fixed in a future version of QLINK.

Caution

Numeric overflow

In cases where you write code which includes values which cause numeric overflow to occur, the assembler truncates the value and generates code for a value within the range of the instruction. A warning of overflow is generated, but these warnings are often ignored. You should verify that warnings will have no effect on your program's performance.

Fixes, Patches, and Workarounds

Fix

BASIC RENUMBER error

An error was noted in the BASIC RENUMBER utility which caused certain strings within quotations to receive renumbering in error. This is a fix to the RENUMBER which was distributed as part of 3.2 PDOS BASIC. The following changes correct this difficulty:

Change:

```
1942  X=FIND: FIND=SRH['"',$LINE[0;X]]: IF FIND<>0: FIND=FIND+X
1943  IF FIND<>0: X=FIND: FIND=SRH['"',$LINE[0;X]]: IF FIND: GOTO 1950
1944  FIND=1
1945  X=FIND: FIND=SRH["'",$LINE[0;X]]: IF FIND<>0: X=FIND+X
1946  IF FIND<>0: FIND=SRH["'",$LINE[0;X]]: IF FIND: GOTO 1950
1947  FIND=1
```

To:

```
1942  X=FIND: FIND=SRH['"',$LINE[0;X]]: IF FIND<>0: X=FIND+X
1943  IF FIND<>0: FIND=SRH['"',$LINE[0;X]]: IF FIND: FIND=X+FIND: GOTO 1950
1944  FIND=1
1945  X=FIND: FIND=SRH["'",$LINE[0;X]]: IF FIND<>0: X=FIND+X
1946  IF FIND<>0: FIND=SRH["'",$LINE[0;X]]: IF FIND: FIND=X+FIND: GOTO 1950
1947  FIND=1
```

Fix

FTELL:C error

An error has been noted in the documentation for the FTELL fix reported in the last volume of *PDOS Tips and Technical Notes.*

The following is incorrect:

```
       asm("beq.s @10");              /* empty buffer */
>      asm("move.w #264(a5),d2");     /* get _iostat for read or write */
       asm("beq.s @20");              /* branch on read flag */
```

The following is correct:

```
       asm("beq.s @10");              /* empty buffer */
>      asm("move.w 264(a5),d2");      /* get _iostat for read or write */
       asm("beq.s @20");              /* branch on read flag */
```

Fix

Parity flag in FBIOSU:SR and F32BIOSU:SR

There is a coding error which prevents FORCE SIO-1 cards from being initialized to even parity. When this is attempted, the system hangs. It may be corrected by using the following procedure.

In file F32BIOSU:SR change the following code in the type 1 baud port procedure

From:

```
       MOVE.B  #$0C,CCR(A0)    ;OUT CLOCK SELECT (DIV BY 2)
       CLR.W   D2              ;ASSUME NO PARITY
       BTST    #BEVP,D1        ;PARITY ENB?
         BEQ.S @004            ;N
       TAS.B    ECR(A0)        ;Y, ENB PARITY GEN/CHECK
*
@004   MOVE.B   #0,ECR(A0)     ;ENABLE/DISABLE PARITY
       CLR.B   TIER(A0)
```

To:

```
       MOVE.B  #$0C,CCR(A0)    ;OUT CLOCK SELECT (DIV BY 2)
*      CLR.W   D2              ;ASSUME NO PARITY
       BTST    #BEVP,D1        ;PARITY ENB?
         BEQ.S @004            ;N
*      TAS.B    ECR(A0)        ;Y, ENB PARITY GEN/CHECK
       MOVE.B   #$80,ECR(A0)
*
@004
*      MOVE.B  #0,ECR(A0)      ;ENABLE/DISABLE PARITY
       CLR.B   TIER(A0)
```

In file FBIOSU:SR also change the following code in the type 2 baud port procedure

From:

```
       MOVE.B  #$0C,SCCR(A0)   ;OUT CLOCK SELECT (DIV BY 2)
       CLR.W   D2              ;ASSUME NO PARITY
       BTST    #BEVP,D1        ;PARITY ENB?
         BEQ.S @004            ;N
       TAS.B    SECR(A0)       ;Y, ENB PARITY GEN/CHECK
*
@004   MOVE.B   #0,SECR(A0)    ;ENABLE/DISABLE PARITY
       CLR.B   STIER(A0)
```

To:

```
       MOVE.B  #$0C,SCCR(A0)   ;OUT CLOCK SELECT (DIV BY 2)
*      CLR.W   D2              ;ASSUME NO PARITY
       BTST    #BEVP,D1        ;PARITY ENB?
         BEQ.S @004            ;N
*      TAS.B    SECR(A0)       ;Y, ENB PARITY GEN/CHECK
       MOVE.B  #$80,SECR(A0)
*
@004
*      MOVE.B  #0,SECR(A0)     ;ENABLE/DISABLE PARITY
       CLR.B   STIER(A0)
```

Fix

BASIC UPTIME utility error

For users of PDOS BASIC, the UPTIME utility contains an error.

Change:

```
530  ON M: ME=31,28,31,30,31,30,29,31,30,31,30,31
```

To:

```
530  ON M: ME=31,28,31,30,31,30,31,31,30,31,30,31
```

Workaround

FORCE ISIO1 card misses re-enable flag

A customer has reported a potential problem with the ISIO1 card from FORCE. When downloading data out a port on the card at high rates, the reenable flag is missed causing the system to hang. It has been indicated that this is a firmware problem which is overcome with delay loops in the BIOS software. A workaround has been suggested which adds a delay loop in the ISIO interrupt service routine under label ISIOHC.

```
@012   MOVE.W  P$ISINT,D1
       BTST    D0,D1          ;TEST HIGH WATER FLAG OF CHANNEL
       BNE.S   @014           ;HIGH WATER IF SET

       MOVE.W  #1200,D0       ;set delay count
@999   DBRA    D0,@999 ;allow ISIO to read int flag as 0

       MOVE.W  #GETCHI,(A0)   ;SET NEW COMMAND

*@014  MOVE.W  D1,P$ISINT     ;(performs no valid function)

@014   MOVE.W  D2,D0          ;GET CHAR TO D0
       AND.W   #$0FF,D0       ;MASK IT
```

Workaround

EQUates in asm instructions in C

To properly utilize equate statements when using the “asm” pseudo-function in a C program, create a file to hold the equate statements and then call this file as part of your asm code sequence.

Save equates in file EQUATE

```
HI    EQU    230
```

In main program include above file

```
main()
{
  asm("include EQUATE");
  asm("move.l  #HI,d1");
}
```

The equate will be properly inserted and resolved when assembled.

```
...
 1  0/00000000:                  test IDNT    5,0
 2                  00012200          OPT ALT,NOWARN,XREF,TC
 3          00000000                  EXTN .main
 4  0/00000000:
 5  0/00000000:                  .main
 6                               *~main:
 7                               *_EnD__=8
 8  0/00000000:4E56FFFC               link.w A6,#-4
 9                               *line 4
10                                    include EQUATE
11
12                  000000E6 HI   EQU     230
13                               *line 5
14  0/00000004:223C000000E6            move.l #HI,d1
15  0/0000000A:                  L1
16  0/0000000A:4E5E                   unlk A6
17  0/0000000C:4E75                   rts
....
```

Workaround

Chaining programs in FORTRAN

Chaining programs in FORTRAN cause problems since the user stack pointer is not updated when using the PDOS XCHF primitive. The reason is that the primitive was designed to pass parameters on the stack to the chained program. To reset the stack pointer the following assembly program may be used to chain programs in FORTRAN and possibly any other language where the stack pointer is not properly located:

```
      XDEF    .CHAIN
*
.CHAIN        XGML                     ;GET MEMORY LIMITS
      MOVEA.L 4(A7),A1        ;GET ADDRESS OF FILENAME
      MOVEA.L EUM$(A6),A7     ;RESET STACK POINTER
      XCHF                     ;CHAIN TO FILE
      XERR                    ;ERROR RETURN TO PDOS
      END
```

Assemble this program with MASM as follows and convert it to a binary file.

```
      xx>MASM CHAIN:SR,#CHAIN:OBJ
      xx>MSYFL CHAIN:OBJ,#CHAIN:SUB
      xx>SA CHAIN:SUB,BN
```

You may include the CHAIN:OBJ into a library if desired or link CHAIN:SUB into your program.

The program call will be as follows:

```
       CHARACTER NULL,FILENAME
       PARAMETER (NULL=0)
       FILENAME='FILENAME:EXT'//NULL        !PDOS requires null termination
        ...
       CALL CHAIN (FILENAME)
```

Another method which may be used to chain to other program and provide a proper stack reset is to use the XEXZ primitive with the new filename as the parameter or command line pointed to by (A1). The code for this primitive is as follows:

```
* XEXZ - EXIT TO MONITOR WITH COMMAND
* INTEGER FUNCTION XEXZ(COMMAND)
* CHARACTER *20 COMMAND
* TYPICAL USAGE MIGHT BE  XEXZ('PROGRAM:SR'//CHAR(0))

        XDEF
.XEXZ   MOVEA.L       4(A7),A1        ;GET THE PASSED PARAMETER
        XEXZ                          ;EXIT TO MONITOR W/ COMMAND
        RTS
        END
```

Workaround

Data offsets exceed displacement in Pascal

In Pascal 3.0 and earlier, if you attempt to access data variables whose address is more than 32k remote, the compiler generates offsets which exceed the word displacement for the register used to point to the variable. The result is an improper access, usually writing the variable over a portion of the program.

Code generated may appear as such:

```
      PEA.L   40000(A4)
```

This should be changed as follows if the displacement is too great:

```
      PEA.L   (A4)           ;push address
      ADDI.L   #40000,(A7)   ;add offset to variable
```

Applications and Hints

Hint

Linking with F77L in Absoft FORTRAN

When linking subprograms with the main program using F77L, there is often not enough space on the line to link all subprograms. Creating a library to include the subprograms can often be time consuming. The following description explains how you might accomplish the link of many subprograms:

F77L may be used serially as many times as you wish on the same main program until all the unresolved references are satisfied. Procedures listed on the command line take precedence over procedures with the same name in library files.

Assume a program A:FOR which calls subroutines B:SUB, C:SUB, D:SUB, E:SUB, and F:SUB. This may be linked as follows:

```
>F77 A              ;Compile main program
>F77L X = A,B,C,D   ;link in 3 subprog and assign to X
>F77L A:PRG = X,E,F ;link in 2 more and assign to A:PRG
>F77L A:PRG,F77:RL/L;link in runtime library if desired
```

The assignment of the program to the temp file X is not required, but saves the original compiled program to prevent the need to recompile it for a later link operation.

If this approach is to be used a number of times until the program is fully debugged, then you can create a PDOS procedure file to execute the compile and link. Any steps not required may be commented out or you may use the AC monitor command to step through to select the start of your procedure file.

Hint

Save time while counting bits

An article recently appeared in EDN magazine that offered a tip on summing the bits in a word. The writer pointed out that it is possible to use the computer's hardware to add up the bits in parallel, rather than adding the bits one at a time. We usually do something like the following:

```
LOOP2
      LSR.W   #1,D3
      BCC.S   @03
      ADDQ.L  #1,D1
@03   DBRA    D5,LOOP2
```

In the above example, D5 serves as the loop counter, D1 is the bit counter, and D3 is the source of the bits to count. This algorithm with appropriate initialization and loops, ran 1677216 iterations in 277.6 seconds. This time may be improved by eliminating the branch and using the ADDX instruction to add the extend bit into D3. You need a dummy data register available which has been initialized to zero. The algorithm is shown below:

```
LOOP2
      LSR.W    #1,D3
      ADDX     D2,D1
      DBRA     D5,LOOP2
```

Data register D2 contains zero and is used as a dummy since there is no instruction to add just the extend bit into D1. This version runs the above number of iterations in 214.8 seconds.

The article pointed out that the ADD instruction will add all the bits together. The article presented the algorithm in pseudo-code which has been converted to 68000 assembly as shown below:

```
*      STEP 1
       MOVE    D0,D1
       MOVE    D0,D2
       LSR     #1,D2
        AND    #$5555,D1
       AND     #$5555,D2
       ADD     D2,D1

*      STEP 2
       MOVE    D1,D2
       LSR     #2,D2
       AND     #$3333,D1
       AND      #$3333,D2
       ADD     D2,D1

*      STEP 3
       MOVE    D1,D2
        LSR    #8,D2
       ADD.B   D2,D1

*      STEP 4
       MOVE    D1,D2
       LSR.B   #4,D2
       AND.B   #$0F,D1
       ADD.B   D2,D1
```

This algorithm uses more code but executes fewer instructions and therefore runs much faster. The 16777216 iterations took only 62.9 seconds.

You might note that doubling the number of bits to check from 16 to 32 would double the time requirements of the first two algorithms, while it would add only one more step in the above example taking perhaps 20% more time.

Application

Multiple program load in Task 0

You may sometimes want to load the code for a program into the parent task's memory and then create small tasks which execute the code. When creating a task using the XCTB primitive, if D0 is zero, then registers A0 and A1 specify the tasks memory limits and A2 specifies the tasks starting PC. This can be used to point tasks into code loaded in the parent task.

To provide a convenient way to resolve the entry points into programs linked into a single module, we wrote a macro program which receives parameter input to define various parameters and then create tasks which will access the previously loaded program.

```
*      PARAMETER FILE TO BE USED WITH MULTIPLE FORTRAN
*      PROGRAMS TO BE LOADED AS ONE PROGRAM
*
*
*      MACRO TO LOAD PARAMETERS FOR TASK
*
*      &2=TASK PORT NUMBER
*      &3=TASK SIZE
*      &4=TASK TIME|PRIORITY
*
*      LOAD NAME,PORT,SIZE,PRIORITY
*
LOAD   MACRO
       XREF     &1
       MOVEQ.L #&3,D0          ;GET TASK SIZE
       MOVEQ.L  #&4,D1         ;GET TASK PRIORITY
       MOVEQ.L #&2,D2          ;GET PORT NUMBER
       LEA.L   KT&#(PC),A2     ;COMMAND LINE POINTER
       XCTB                    ;CREATE TEMP TASK TO FREE MEM
         BNE.S ERR&#
       MOVE.L  #100,D0
        MOVE.L #128,D1         ;
       XDEV
       XSUI
       MOVEQ.L  #&3,D0         ;GET TASK SIZE
       MOVEQ.L #&4,D1          ;GET TASK PRIORITY
       MOVEQ.L #&2,D2          ;GET PORT NUMBER

       XGUM                    ;GET MEMORY FOR TASK
         BNE.S  ERR&#
       LEA.L   &1(PC),A2       ;GET ENTRY ADDRESS OF PROGRAM
       MOVE.L  #0,D0           ;USE MEMORY BOUNDS
       XCTB                    ;CREATE TASK AND GO THERE
         BNE.S  ERR&#
       BRA.S   X&#             ;BRANCH AROUND VARIABLES
*
ERR&# XERR
KT&#   DC.B    'LT.KT',0
       EVEN
X&#    NOP
       ENDM
```

Multiple Program Load in Task 0 (cont'd)

A LOAD:SR file is created to pass the parameters to the macro.

```
      OPT PDOS
      INCLUDE LOAD:MAC
START LOAD PROG1,2,30,64
      LOAD PROG2,4,30,64
      LOAD PROG3,3,30,64
      LOAD PROG4,5,30,64
      XEXT
      END
```

LOAD:SR is assembled to the file LOAD:OBJ and is linked with the programs as follows:

```
QLINK
Z
BASE,$5D00                ;EPROM or Program load address
SECTION 0,$5D00
IN LOAD:OBJ               ;Loader module
EVEN
DEFINE PROG1,Q$H0         ;Define link address
IN PROG1:OBJ              ;Load program 1
DEFINE PROG2,Q$H0         ;Define link address
IN PROG2:OBJ              ;Load program 2
DEFINE PROG3,Q$H0         ;Define program link
IN PROG3:OBJ              ;Load program 3
DEFINE PROG4,Q$H0         ;Define program link
IN PROG4                  ;Load program 4
SY                        ;Define as SY file
OUT #LOAD                 ;Output to LOAD program
MAP ALL #LOAD:MAP         ;Output map of linkage
END
Q                         ;Quit
```

When the program LOAD is executed, it loads all of the linked code, creates tasks for the programs to run and sets up pointers into the loaded code to execute the programs.

This application could allow several tasks to access common code by using the same entry point for each task.