# SYS68K/PDOS

Product Overview

First Edition
April 1985

FORCE COMPUTERS Inc./GmbH
All Rights Reserved

PART NO. 800030

## NOTE

FORCE Computers Inc.
727 University Avenue
Los Gatos, CA 95030
U.S.A.

Phone : (408) 354 34 10
Telex : 172465
FAX : (408) 395 77 18

FORCE COMPUTERS GmbH
Daimlerstrasse 9
D-8012 Ottobrunn/Munich
West-Germany

Phone : (089)6 09 20 33
Telex : 5 24 190 forc-d
FAX : (0 89)6 09 77 93

FORCE COMPUTERS FRANCE Sarl
11, rue Casteja
92100 Boulogne
France

Phone : (1) 620 37 37
Telex : 206 304 forc-f
FAX : 1 621 35 19

# PDOS DISK OPERATING SYSTEM

## PRODUCT OVERVIEW

### INDEX OF CONTENTS

FEATURES:

-REAL-TIME, MULTI-USER, MULTI-TASKING

-PRIORITIZED, ROUND-ROBIN SCHEDULING

-INTERTASK COMMUNICATION AND SYNCHRONIZATION

-TASK MEMORY MAP CONTROL FOR PROGRAM SECURITY

-FULL EXCEPTION PROCESSING

-SEQUENTIAL, RANDOM, AND SHARED FILE MANAGEMENT

-HARDWARE INDEPENDANCE

-68000 LAYERED DESIGN OF KERNEL, FILE MANAGER, MONITOR

-COMPLETE FLOATING POINT SUPPORT

-CONFIGURABLE, MODULAR, ROMABLE STANDALONE SUPPORT

-NO MEMORY RESTRICTIONS

---

## 1.DESCRIPTION:

PDOS* is a powerful multi-user, multi-tasking operating system developed for the 32-bit Motorola 68ØØØ processor family. This development software is designed for scientific, educational, industrial, and business applications.

PDOS* consists of a small, real-time, multi-tasking kernel layered by file management, floating point, and user monitor modules. The 2k byte kernel provides synchronization and control of events occurring in a real-time environment using semaphores, events, messages, mailboxes, and suspension primitives. All user console I/O as well as other useful conversion and housekeeping routines are included in the PDOS* kernel.

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

Assembly language floating point applications are no longer a problem. Conversion modules, assembler directives, and operating system calls allow easy integration of floating point operations into user application programs.

## 2.FUNCTIONAL DESCRIPTION:

PDOS* KERNEL. PDOS* is written in 68ØØØ assembly language for fast, efficient execution. The small kernel provides multi-tasking, real-time clock, event processing, and memory management functions. Ready tasks are scheduled using a prioritized, round-robin method. Three XOP vectors are used to interface over 75 system primitives to a user task.

MULTI-TASKING EXECUTION ENVIRONMENT. Tasks are the components comprising a real-time application. Each task is an independant program that shares the processor with other tasks in the system. Tasks provide a mechanism that allows a complicated application to be subdivided into several independant, understandable, and manageable modules. Real-time, concurrent tasks are allocated in 2k byte increments. Task system overhead is less than 1k bytes.

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

MEMORY REQUIREMENTS. PDOS* is very memory efficient. The PDOS* kernel, floating point module, file manager, and user monitor utilities require only 8k bytes of memory plus an additional 4k bytes for system buffers and stacks. Most applications can be developed and implemented on the target system. Further memory reduction can be achieved by linking the user application to a 2k byte PDOS* kernel for a small, ROMable, standalone, multi-tasking module. A fast, 6k byte scientific orientated BASIC interpreter with real-time primitives provides interactive high level language support as well. For large system configurations, PDOS* effectively addresses up to a 32 bit address space.

FILE MANAGEMENT. The PDOS* file management module provides sequential, random, read only, and shared access to named files on a secondary storage device. These low overhead file primitives use a linked, random access file structure and a logical sector bit map for allocation of secondary storage. No file compaction is ever required. Files are time stamped with date of creation and last update. Up to 32 files can be simultaneously opened. Complete device independence is achieved through read and write logical sector primitives.

COMMAND LINE INTERPRETER. A resident command line interpreter allows multiple commands to be enterred on a single line. Command utilities such as append, define, delete, copy, rename, and show file are also resident and can be executed without destroying current memory programs. Other functions resident in the monitor include setting the baud rate of a port, checksumming memory, creating tasks, listing tasks, files and open file status, asking for help, setting file level, file attributes, interrupt mask, and system disk, and directing console output.

INTERRUPT MANAGEMENT. The PDOS* kernel handles user console, system clock, and other designated hardware interrupts. User consoles have interrupt driven character I/O with type ahead. A task can be suspended pending a hardware or software event. PDOS* will switch control to a task suspended on an external event within 1ØØ microseconds after the occurrence of the event (provided the system mask is high enough.) Otherwise, a prioritized, round-robin scheduling of ready tasks occurs at 1Ø millisecond intervals.

PORTABILITY. PDOS* gives software portability through hardware independance of read/write logical sector primitives. All other hardware functions such as clocks, mappers, and UARTS are conveniently isolated for minimal customization to new 68000 based systems.

CUSTOMER SUPPORT. Numerous support utilities including virtual screen editors, assembler , linker, macroprocessor, disk diagnostics, link, and recovery, disk cataloging are standard. Single stepping, multiple break points, memory snap shots, save and restore task commands, and error trapping primitives are provided in all languages to aid in program debugging.

## 3.LANGUAGE SUPPORT:

| | |
|---|---|
| -Basic | Standard Dartmouth Basic with enhancements, such as program debugging, inter-task communication and real-time support. |
| -Pascal | multi-pass, optimizing compiler that generates assembler text for the 68000 microprocessor. The PDOS* Pascal compiler implements a superset of the Pascal language defined by Jensen and Wirth. |
| -Fortran 77 compiler, supporting the full ANS Fortran 77 standard | |
| -C | compiler for the c language |

## PDOS KERNEL

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

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

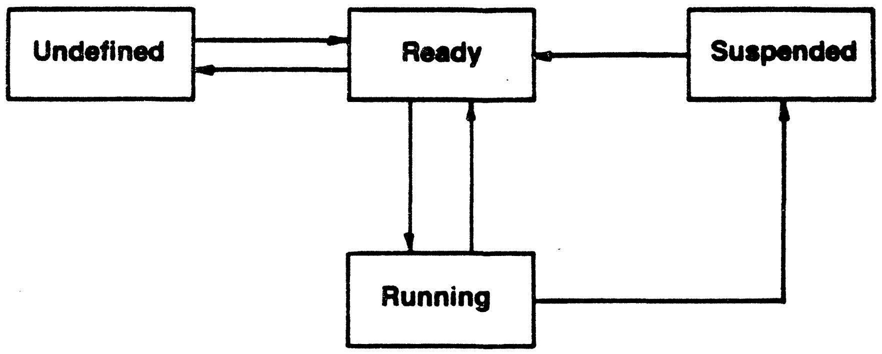
## PDOS TASK

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

The task list is used by the PDOS kernel to schedule tasks. A task list entry consists of a priority, task time, task number, task control block pointer, task map constant, and two suspended event registers.

The first 500 hex bytes of a task is the task control block. This block of memory consists of three buffers and parameters peculiar to the task. The 68000 address register A6 points to the status block when the user program space is entered.

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



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

1. Undefined--A task is in this state before it is loaded into the task list. It can be a block of code in a disk file or stored in memory.

2. Ready-- When a task is loaded in memory and entered in the task list but not executing, it is said to be ready.

3. Running-- A task is executed when scheduled by the PDOS kernel from the task list.

4. Suspended-- When a task is stopped pending an event external to the task, it is suspended. A suspended task moves to the ready or running state when the event occurs.

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

## MULTI-TASKING

Up to 32 independent tasks can reside in memory and share CPU cycles. Each task contains its own task control block and thus executes independantly of any other task. A task control block consists of buffers, pointers, and a PDOS scratch area.

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

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

2. The task memory requirement in 2K byte increments.

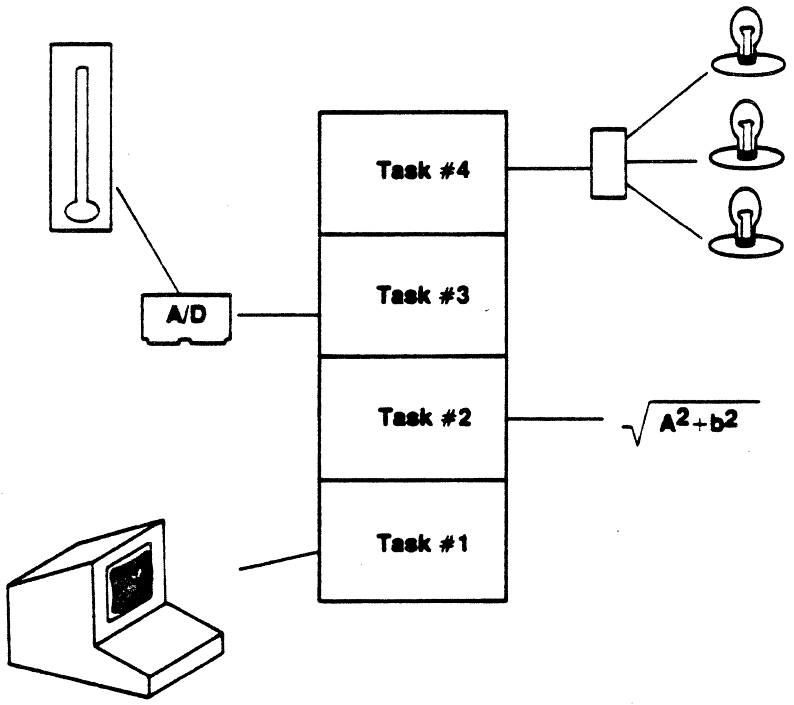
3. An input/output port for task console communication.

4. A task command.

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

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

A task entry in the task list queue consists of a task number designation, parent task number, time interval memory page constant, task control block pointer, and two event registers. Swapping from one task to the next is done when the task interval time decrements to zero or during an I/O call to PDOS. The task interval timer decrements by one every ten milliseconds.



Any task may spawn another task. Memory for the new task is allocated in 2K byte blocks from a pool of available memory. If no memory is free, the spawning task's own memory is used and the parent task's memory is reduced in size by the amout of memory allocated to the new task.

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

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

## PDOS CHARACTER I/O

The flow of character data through PDOS is the most visible function of the operating system. Character buffering or type-ahead assures the user that each keyboard entry is logged, even when the application is not looking for characters. Character output is normally through program control (polled I/O). However, an interrupt driven output primitive allows maximum data transfer even though the task itself may be in a ready or suspended state.

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

### PDOS CHARACTER INPUTS

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

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

If IMP$(A6) is zero and ACI$(A6) is nonzero, then a request is made to the file manager to read one character from the file assigned to ACI$(A6). The character then comes from a disk file or an I/O device driver. This continues until an error occurs (such as an END-OF-FILE) at which time the file is closed and ACI$(A6) is cleared.
If both IMP$(A6) and ACI$(A6) are zero, then the logical input port buffer selected by PRT$(A6), is checked for a character. If the buffer is empty, then the task is automatically suspended until a character interrupt occurs.

## PDOS CHARACTER OUTPUTS

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

When an output primitive is called, the task output unit is ANDed with the task spooling output unit. If the result is nonzero, then the character is directed to the file manager and written to the file specified by SFI$(A6). The output unit is then masked with the complement of the spooling unit and passed to the unit 1 and unit 2 processors.

Units 1 and 2 are special output numbers. Unit 1 is the console output port assigned when the task was created. Unit 2 is an optional output port that is assigned by the user task in addition to unit 1. Unit 2 is set by the baud port commands.

If the 1. bit (LSB) is set in the masked output unit, then the character is directed to port U1P$(A6). Likewise, if the 2. bit is set in the masked output unit, then the character is output to the U2P$(A6) port.

In summary, the bit positions of the output unit are used to direct output to various destinations. More than one destination can be specified. Bits 1 and 2 are predefined according to U1P$(A6) and U2P$(A6) variables within the task control block. Other unit bits are used for outputs to files and device drivers. Thus, if SPU$(A6)=4 and UNT$(A6)=7, then output would be directed to the file manager via SFI$(A6) and to two UARTS as specified in U1P$(A6) and U2P$(6).

```
SPU$(A6) = ØØØØ ØØØØ ØØØØ Ø1ØØ

UNT$(A6) = ØØØØ ØØØØ ØØØØ Ø111
                             |||
                             |||
                             |||
     File SFI$(A6)----------'||
                              ||
     Port U2P$(A6)-----------'|
                               |
     Port U1P$(A6)------------'
```

## EVENTS

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

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

1. Events 1 through 7 are hardware events. They correspond to interrupt levels 1 through 7 of the MC68ØØØ CPU. When a task suspends itself pending a hardware event, the system mask should be enabled allowing the interrupt to occur. When the interrupt does occur, the corresponding event bit is set, the system mask is raised to block further interrupts by that level, and a task swap is initialized. If the current task has not locked itself in the execution state, then the highest priority ready task is awakened, swapped in, and begins executing. It is the responsibility of the awakened task to acknowledge the interrupt (puts its hand down) and then lower the system mask.

2. Events 8 through 63 are software events. They are set and reset by tasks and not changed by any PDOS system function. A task can suspend itself pending a software event and then be rescheduled when the event is set. One task must take the responsibility of resetting the event for the sequence to occur again.

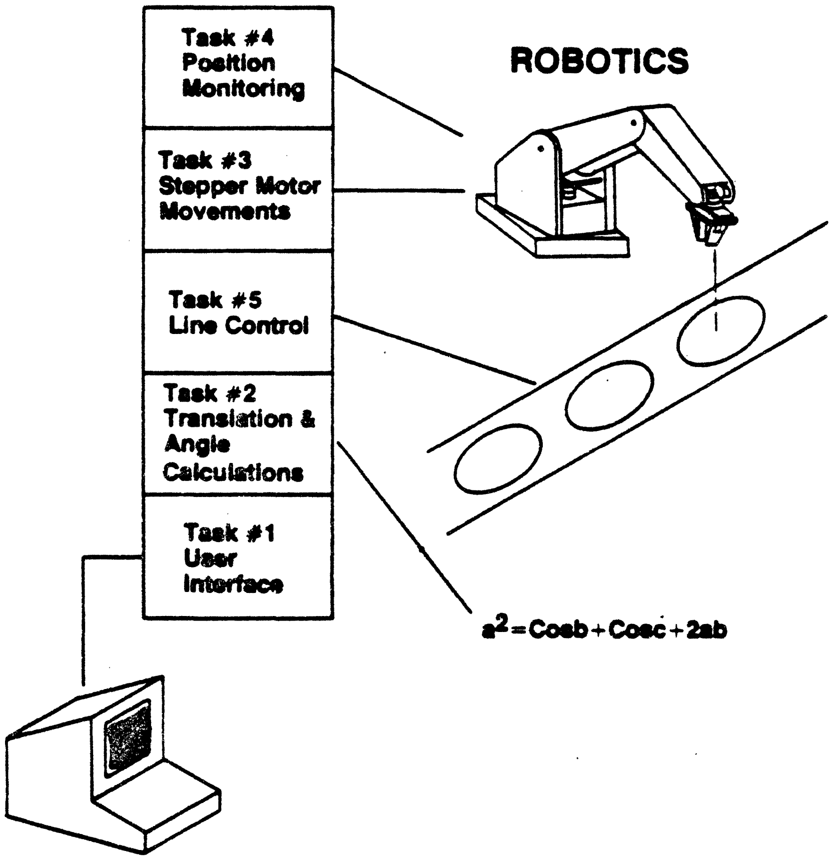
3. Events 64 through 95 are like the normal software events except that PDOS resets the event whenever a task suspended on that event is rescheduled. Thus, only one task is rescheduled when the event occurs.

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

5. Events 112 through 115 are timing events and are set automatically by the PDOS clock module according to intervals defined in the PDOS Basic I/O module (BIOS). They are altered at run time by the BFIX utility. Event 112 is measured in tics, while events 113, 114, and 115 are in seconds. The maximum time interval for event 112 is 497 days. Events 113, 114, and 115 have a maximum interval of 4, 297, 967, 3ØØ seconds or approximately 136 years. A task suspended on one of these events is regularly scheduled on a tic or second boundary.

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

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


Task #4 Position Monitoring
Task #3 Stepper Motor Movements
Task #5 Line Control
Task #2 Translation & Angle Calculations
Task #1 User Interface
ROBOTICS
a2=Cosb+Cosc+2ab

## TASK COMMUNICATION

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

### MAIL array

The MAIL array is a permanent 254 byte memory buffer accessible by assembly language programs and PDOS BASIC as the singly dimensioned array MAIL [Ø] through MAIL [3Ø]. This array is never cleared even during PDOS initialization.

### COM array

The COM array (COMmon array ) is a singly dimensioned array which is used by PDOS BASIC to preserve data during RUN, NEW, and program chaining. In addition, COM is used to pass and return parameters to assembly language subroutines. The COM array is defined within each task and is neither permanent nor resident at a fixed memory address.

### Absolute data movement

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

### Event flags

Event flags are global system memory bits, common to all tasks. They are used in connection with task suspension or other mailbox functions. Events are discussed in detail in the previous section.

### Message buffers

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

Memory Mailbox

The utilities ALOAD and FREE are used to permanently allocate system memory for non-tasking data or program storage. Memory allocated in this way can be used for mailbox buffers as well as handshaking semaphores or assembly programs.

## TASK SUSPENSION

Any task can be suspended pending one or two hardware or software events. A suspended task does not receive any CPU cycles until one of the desired events occurs. A task is suspended from BASIC by using the WAIT command, or from an assembly language program by the XSUI primitive. A suspended task is indicated in the LIST TASK (LT) command by the event number(s) being listed under the 'EVENT' heading.

When one of the events occurs, the task is rescheduled and resumes execution. If the event is a hardware interrupt (events 1 through 7), then the corresponding event is set and an immediate swap occurs. If a high priority task is waiting for the event, it is immediately rescheduled, overriding any current task (unless locked). If the event is a software event (8 through 128), then the task begins execution during the normal swapping function of PDOS.

## INTERRUPTS

PDOS supports user interrupt routines for levels 1, 2, and 3 or as defined by the Basic I/O (BIOS) module.

## PDOS FILE MANAGEMENT

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

## PDOS FILE STORAGE

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

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

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

A PDOS file is accessed through an I/O channel called a file slot. Each file slot consists of a 32 byte status area and an associated 256 byte sector buffer. Data movement is always to and from the sector buffer according to a file pointer maintained in the status area. Any reference to data putside the sector buffer requires the buffer to be written to the disk (if it was altered) and the new sector to be read into the buffer. The file manager maintains current file information in the file slot status area such as the file pointer, current sector in memory, END-OF-FILE sector number, buffer in memory flag, and other critical disk parameters required for program-file interaction.

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

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

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

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

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

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

## FILE NAMES

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

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

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

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

## DIRECTORY LEVELS

Each PDOS disk directory is partitioned into 256 directory levels. Each file resides on a specific level, which facilitates selected directory listings. You might put system commands on level Ø, procedure files on level 1, object files on level 1Ø, listing files on level 11, and source files on level 2Ø.

PDOS operates in a soft or hard partition mode. In soft partition mode, all files are global with respect to a disk directory and can be accessed without referencing the file level. File names are not unique to a level, hence the same file name cannot be used twice in any one disk directory.

In hard partition mode, each directory level is unique with the exception of level Ø which remains global. Duplicate file names can exist on the same disk on different levels.

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

## DISK NUMBERS

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

The range of disk numbers is from Ø to 255. Several disk numbers may share the same secondary storage device. Each disk can have a maximum of 65282 sectors or 16,711,68Ø bytes.

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

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

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

## FILE ATTRIBUTES

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

The file type is used by the PDOS monitor in processing the file. For instance, a file typed as 'EX' (a PDOS BASIC file), invokes the BASIC interpreter, loads the file, and begins execution with the first line number. A file typed as 'OB' (a 68ØØØ object module), is passed to a relocating loader and loaded into memory. If a start address tag is included at the end of the file, the module is immediately executed.

The following are legal PDOS file types:

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

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

OB — 68ØØØ tag object file. All assembly commands are typed as object files. This directs the PDOS monitor to load the file into memory and execute the program.

SY — System file. A 'SY' file is generated from an 'OB' file. MC68ØØØ object is condensed into a smaller and faster loading format by the 'SYFILE' utility.

BX — PDOS BASIC binary file. A BASIC program stored using the 'SAVEB' command is written to a file in pseudo-source token format. Such a file requires less memory than the ASCII LIST format and loads much faster. Subsequent reference to the file name via the PDOS monitor automatically restores the tokens for the BS4ASIC interpreter and begins execution.

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

TX — ASCII text file. A 'TX' type classifies a file as containing ASCII character text. Reference to the file name via the PDOS monitor causes the file to be listed to your console.

DR — I/O driver. A 'DR' file type indicates that the file data is an I/O driver program and is executed when referenced.

A PDOS file is physically stored in contiguous or non-contiguous sectors depending upon how it was initially created. Contiguous files have random access times far superior to non-contiguous files. A contiguous file is indicated in the directory listing by the letter 'C' following the file type.

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

File storage method and protection flags are summarized as follows:

> C — Contiguous file. A contiguous file is organized on the disk with all sectors logically sequential and ordered. Random access in a contiguous file is much faster than in a non-contiguous file since the forward/backward links are not required for positioning.
>
> * — Delete protect. A file which has one asterisk as an attribute cannot be deleted from the disk until the attribute has been changed.
>
> ** — Delete and write protect. A file which has two asterisks as an attribute cannot be deleted nor written to. Hence READ, POSITION, REWIND, and CLOSE are the only legal file operations.

## FLOATING POINT MODULE

The PDOS floating point module is a single accumulator, IBM excess 64 format, multi-user floating point processor. It includes all the necessary routines to write assembly language floating point software and supports the PDOS BASIC interpreter.

Floating point commands are referenced using the F-line ($FØØØ) exception instructions of the 68ØØØ. Parameters are passed in address register AØ.

Commands include the following:

| Symbol | Value | | Description |
|---|---|---|---|
| FLDD. | $FØØØ | = | Load FPA |
| FSRD. | $FØØ2 | = | Store FPA |
| FADD. | $FØØ4 | = | Add to FPA |
| FSUB. | $FØØ6 | = | Subtract from FPA |
| FMUL. | $FØØ8 | = | Multiply FPA |
| FDIV. | $FØØA | = | Divide into FPA |
| FSCL. | $FØØC | = | Scale FPA |
| FCLR. | $FØØE | = | Clear FPA |
| FFLT. | $FØ1Ø | = | Float FPA |
| FNRM. | $FØ12 | = | Normalize FPA |
| FNEG. | $FØ14 | = | Negate FPA |
| FABS. | $FØ16 | = | Absolute value of FPA |
| FPST. | $FØ18 | = | Read FPA status |
| FTIC. | $FØ1A | = | Load clock TICS. |
| FINV. | $FØ1C | = | Invert FPA |
| FELD. | $FØ1E | = | Load error register address |

## PDOS RESIDENT MONITOR COMMANDS

| | | |
|---|---|---|
| AF—Append file | GO—Execute | RD—RAM disk |
| AM—Available memory | HE—Help | RN—Rename file |
| BP—Baud port | ID—Init date and time | RS—Reset |
| CF—Copy file | IF—Conditional | SA—Set file attributes |
| CT—Create task | IM—Interrupt mask | SF—Show file |
| DF—Define file | KM—Kill message | SM—Send task message |
| DL—Delete file | KT—Kill task | SP—Disk usage |
| DM—Delete multiple | LO—Load file | SU—Spool unit |
| DT—Display time | LS—List directory | SY—System disk |
| EV—Events | LT—List tasks | TM—Transparent mode |
| EX—Basic | LV—Directory level | TP—Task priority |
| FM—Free memory | MF—Make file | UN—Output unit |
| FS—File slots | PB—Debugger | ZM—Zero memory |
| GM—Get memory | RC—Reset console | |

## MONITOR COMMAND FORMATS

```
AF <file1>,<file2>                 LO <file>
AM                                 LS {<list>}
BP <prtt>,<rat>{,<typ>,<bas>}      LT
CF <file1>,<file2>                 LV {<level>}
CT <cmd>,<sze>,<prity>,<prt>       MF {<file>}
DF <file>{,<size>}                 PB
DL <file>                          RC
DM <filelist>                      RD{<unt>,<sze>,<adr>}
DT                                 RN <file1>,<file2>
EV {<event>}                       RS {<disk>}
EX                                 SA <file>{,<attribute>}
FM <kbytes>                        SF {-}<file>
FS                                 SM{<task#>,<message>}
GM {<kbytes>}                      SP {<disk>}
GO {<address>}                     SU <unit>{,<file>}
HE {<list>}                        SY {<disk>}
ID                                 TM {<port>}{,<break>}
IF <str1>{=<str2>}                 TP {<task#>,}<proirity>
IM <mask>                          UN <unit>
KM <task#>                         ZM
KT {-}<task#>
```

## PDOS UTILITIES

A PDOS utility is an auxiliary program that resides on the disk. Written in either assembly language or BASIC, PDOS utilities are run by simply entering the name of the desired utility. Of course, the utility must be a file on the disk with the appropriate attributes. Following is a list and a brief description of each utility.

PDOS* UTILITIES

| | |
|---|---|
| MASM | 68000 assembler. |
| MBACK | Disk backup. |
| BXREF | Basic cross reference. |
| COMP | Compare ASCII files. |
| MCHATLE | Changes attributes and levels of selected files. |
| MDDMAP | Disk diagnostic. Reads files by links. |
| MDDUMP | Disk sector dump and alter. |
| MDISCAT | Catologues combined directories of multiple disks. |
| MDNAME | Renames PDOS disks. |
| MFDUMP | Output logical dump of PDOS files. |
| FFRMT | Format logical unit |
| MFSAVE | Restore files from links. |
| MINIT | Initialize PDOS disk. |
| MLDIR | Wild card list directory. |
| MLEVEL | Short listing by level. |
| LIBGEN | Create user module library. |
| QLINK | Link relocatable object. |
| MORDIR | Alphabetizes and compresses disk directory. |
| SYFILE | Generate SY file from OB. |
| MTERM | Set terminal cursor functions for task only. |
| MTRANS | Selective file transfers. |
| RENUMBER | Renumbers BASIC programs. |
| UPTIME | System uptime |

## PDOS ASSEMBLY PRIMITIVES

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

PDOS calls are divided into three categories; namely, 1) system, 2) console I/O, 3) files.

### SYSTEM CALLS

| | |
|---|---|
| XBUG—Debug call | XLER—Load error register |
| XCBD—Convert binary to decimal | XLKT—Lock task |
| XCBH—Convert binary to hex | XLSR—Load status register |
| XCBM—Convert to decimal with message | XRDM—Dump registers |
| XCBX—Convert to decimal in buffer | XRDT—Read date |
| XCDB—Convert decimal to binary | XRTM—Read time |
| XCHX—Convert binary to hex in decimal | XRTS—Read task status |
| XCTB—Create task block | XSEF—Set event flag |
| XDEV—Delay set/reset event | XSTM—Send task message |
| XDTV—Define trap vectors | XSTP—Set/read task priority |
| XERR—Return error to do monitor | XSUI—Suspend until interrupt |
| XEXC—Execute PDOS call D7.W | XSUP—Enter supervisor mode |
| XEXT—Exit to monitor | XSWP—Swap to next task |
| XFTD—Fiw time and date | XTEF—Test event flag |
| XFUM—Free user memory | XUDT—Unpack date |
| XGNP—Get next parameter | XULT—Unlock task |
| XGTM—Get task message | XUTM—Unpack time |
| XGUM—Get user memory | XWDT—Write date |
| XKTB—Kill task | XWTM—Write time |
| XKTM—Kill task message | |

### CONSOLE I/O PRIMITIVES

| | |
|---|---|
| XBCP—Baud console port | XPCL—Put CRLF |
| XCBC—Check for break character | XPDC—Put data to console |
| XCBP—Check for break or pause | XPEM—Put encoded message to console |
| XCLS—Clear screen | XPLC—Put line to console |
| XGCC—Get character conditional | XPMC—Put message to console |
| XGCR—Get character | XPSC—Position cursor |
| XGLB—Get line in buffer | XPSP—Put space to console |
| XGLM—Get line in monitor buffer | XRCP—Read port cursor position |
| XGLU—Get line in user buffer | XRPS—Read port status |
| XGML—Get memory limits | XSPF—Set port flag |
| XPBC—Put buffer to console | XTAB—Tab to column |
| XPCC—Put character(s) to console | |

## FILE PRIMITIVES

| | |
|---|---|
| XAPF—Append file | XRDE—Read next directory entry |
| XBFL—Build file directory list | XRDN—Read directory entry by name |
| XCFA—Close file with attribute | XRFA—Read file attributes |
| XCHF—Chain command | XRLF—Read line from file |
| XCLF—Close file | XRNF—Rename file |
| XCPY—Copy file | XROO—Open random read only |
| XDFL—Define file | XROP—Open random |
| XDLF—Delete file | XRSE—Read sector |
| XFBF—Flush buffers | XRST—Reset disk |
| XFFN—Fix file name | XRSZ—Read sector zero |
| XISE—Initialize sector | XRWF—Rewind file |
| XLDF—Load file | XSOP—Open sequential |
| XLFN—Look for name in file slots | XSZF—Get disk file |
| XLKF—Lock file | XULF—Unlock file |
| XLST—List file directory | XWBF—Write bytes from file |
| XNOP—Open non-exclusive random | XWFA—Write file attributes |
| XPSF—Position file | XWLF—Write line from file |
| XRBF—Read bytes from file | XWSE—Write sector |
| XRCN—Reset console inputs | XZFL—Zero file |

Program Example for the PDOS 68ØØØ Assembler:

```
START     MOVEQ.L #Ø,D1          ; GET DEFAULT
          XPMC MES1              ; OUTPUT HEADER
          XGLU                   ; GET REPLY
            BLS.L STRTØ2         ; USE DEFAULT
          XCDB                   ; CONVERT, O.K. ?
            BGT.S STRTØ2         ; YES
          XPMC ERM1              ; NO, REPORT ERROR
          BRA.S START            ; TRY AGAIN
*
*
STRTØ1    MOVE.L D1,D5           ; SAVE VALUE
          ......

MES1      DC.B $ØD,$ØA,'ANSWER = ',Ø
ERM1      DC.B $ØD,$ØA,'INVALID !',Ø
          EVEN
```

## PDOS PARALLEL PASCAL

The following are some of the major features of PDOS Pascal:

PDOS Pascal runs on all 68ØØØ PDOS systems.

The PDOS Pascal compiler generates assembler text (not p-code)

PDOS Pascal applications are designed for process control, instrumentation, automation, robotics, CAD/CAM, and real-time operations.

PDOS Pascal performs both single and double precision operations for real numbers. Single precision is accurate to 6.5 decimal places and double precision is accurate to 15.5

PDOS Pascal allows integer length to be declared to one, two, or four bytes. This aids in faster calculation.

PDOS Pascal is inherently modular and aids the programmer in designing block-structured code.

PDOS Pascal has the capacity for virtually unlimited concurrent tasks.

PDOS Pascal procedures can be designated as EXTERNAL and compiled seperately. This is especially important for complex software development.

PDOS Pascal is geared towards experienced Pascal programmers. Type checking is relaxed to allow for systems programming.

## DESCRIPTION:

PDOS PASCAL is a modern, multiple-pass, optimizing compiler that gererates assembler text for the MC68ØØØ microprocessor instruction set. The PDOS PASCAL compiler implements a superset of the Pascal language defined by Jensen and Wirth that includes extentions for writing multiple task programs for concurrent programming. This capability makes PDOS PASCAL ideal for process control, instrumentation, automation, robotics, CAD/CAM, and numerous other applications requiring real-time response and interrupt handling.

PDOS PASCAL is designed to enhance the PDOS operating system as a resident development tool for real-time applications. The source code text output by the compiler can be either edited or assembled into object code for linking or running under PDOS. All compiler output is ROMable for stand-alone applications. The system includes the compiler, code generator, run-time library in object form, utilities, and sources to selected run-time library modules.

PDOS PASCAL was specifically designed to handle your real-time processing needs. The extensions make PDOS PASCAL much more than just an ordinary general-purpose software product, by providing you with additional language constructs that are useful for writing multiple-task programs and handling system interrupts.

The PDOS PASCAL language structure lets you seperately define each process or task and interrupt service to be handled. The language structure also provides inter-process communication through the use of an additional data type called SIGNAL. With SIGNAL, task synchronization is possible via user-written semaphores.

PDOS PASCAL was designed to be modular, so that applications can be developed one piece at a time. System libraries can be built as modules are created. Interface to the library modules can be via PDOS PASCAL EXTERNAL calls. Library modules are then loaded at system link time. In addition, READ and WRITE have been extended to work with procedures.

## FUNCTIONAL DESCRIPTION:

PASCAL DATA TYPES. PDOS PASCAL supports Integer, Real, Boolean, and Char data types. Additionally, a data type can be defined as a subrange of an ordinary type (integer, boolean, char, or enumeration type) in which the least and largest values of the subrange are identified. An Array type is a structure consisting of a fixed number of components all of the same type, called the component type, in which the elements of the array are designated in indicies. The array-type definition specifies the component type and the index type. Component type may be any type including another standard type.

The record data type consists of a fixed number of components that can be of different types. For each component, called a field, the record definition specifies its type and identifier. Set type defines the range of values that is the powerset of a base type, which can be integer, boolean, char, or subrange or any enumeration type. File type defines a structure consisting of a sequence of components all of the same type. The number of components (length) of the file is not fixed by the file definition.

PASCAL EXTENTIONS. The parallel-processing features of MODULAR have been included in PDOS PASCAL. They include PROCESS,SEND, and WAIT, and the data type SIGNAL. READ and WRITE procedures can accept as their first parameter the name of a user-written procedure. Input or output is directed through this procedure instead of to a file.

Procedures can be designated EXTERNAL, compiled by themselves and added to the program at link time. The word-symbol ORIGIN is used to locate a variable at a fixed memory location for interrogating hardware-device registers. XOR, exclusive OR, is present and the NOT,OR,AND,and XOR operators have been extended to operate on integers as well as booleans.

The CASE statement has an OTHERWISE clause, the CLOSE procedure closes a file, and a SEEK procedure exists, allowing random access to file elements. The FLOAT function (converting integer to real) is available explicitly. RESET and REWRITE procedures have optional second, third and fourth arguments to specify the name and size of a file.

Declaration of a procedure or function parameter must include a dummy parameter list. This feature, from the ISO draft for PASCAL, allows the compiler to check the types of the parameters when the formal procedure is called.

Underscore "_" may be used within an identifier for clarity. However, it is not considered part of the identifier. For example, FIRST_ONE is recognized as the same as FIRSTONE. Underscore may not appear in a word-symbol, e.g., A_ND is not recognized as AND.

The declaration sections for labels, constants, types, and variables may occur in any order. They must precede procedure declarations. Every name or label must be declared before it is used.

PASCAL RESTRICTIONS. The PACK, UNPACK, and PAGE procedures are not implemented. The transcedental math functions SIN, COS, ARCTAN, LN, and EXP and SQRT are not recognized by the compiler. Pascal source-code versions of these functions are provided and can be included in the user's program. The NEW and DISPOSE procedures do not use variant tag fields. Standard functions and procedures cannot be used as function or procedure parameters. INPUT and OUTPUT are not predefined as the default input and output files, but rather to and from the system terminal. Single-pass scope rules are followed. Type checking of subranges is relaxed. Conformant array parameters are not implemented.

IMPLEMENTATION LIMITS. Identifiers are recognized by their first 1Ø characters; the rest are scanned but ignored. Labels consist of up to 4 digits. Lower-case letters and the corresponding upper-case letters are recognized as the same characters. Lower-case letters in strings and comments are untouched. User-defined enumeration types may not have more than 256 members. Lines of source text must be less than 132 characters long. Strings are limited to 8Ø characters and Sets can contain a maximum of 96 elements. WITH statements can be nested up to 12 deep.

PDOS PASCAL Example Program:

```
program RAINFALL(input,output);
     { taken from PASCAL PROGRAMMING STRUCTURES }
     {            FOR MOTOROLA MICROPROCESSORS  }
     {            GEORGE W. CHERRY              }
type
   RainfallType = array[1..12] of real;
var
   Month      : 1..12;
   Rainfall   : RainfallType;
   Sum,Average : real;
procedure SortRainfall;
     begin
        writeln('SortRainfall')
     end;
begin
     Sum :=Ø;
     writeln;  { new line  }
     writeln;
     for Month :=1 to 12 do
       begin
         write('Enter rainfall for month',Month: 3 ,'    :');
         read(Rainfall[Month]);
         write(CHR(16#ØD));
         write ('                                          ');
         write(CHR(16#ØD));
         Sum := Sum + Rainfall[month]
       end;
   writeln;
   writeln;
   Average := Sum/12;
   writeln('The monthly average is ',Average);
   writeln;
   writeln('The deviation from the average for');
      for Month :=1 to 12 do
         writeln('                    month ',Month: 3, ' :  ',
                  Rainfall[Month] - Average);
     SortRainfall
  end.
```

# 68ØØØ PDOS BASIC

## FEATURES

- Meaningful, unlimited length variable names
- Multiple line, recursive functions
- Local function variables
- Multi-dimensioned arrays
- Extensive line editing commands
- Fast 64-bit floating point arithmetic
- Context oriented string commands
- Full disk file interface commands
- Transfers and subroutine calls to labels
- Standalone run module support
- No 64K byte boundary restrictions
- Assembly language loader and linkage
- Variable, transfer, and execution trace
- Program chaining
- Formatted print commands
- Intertask communication arrays
- Logical and Boolean operators
- Time and date commands
- Set and test event commands
- Suspend task command

## DESCRIPTION:

Microcomputer interpreters are generally slow and not competetive in performance with similar compilers. Despite this disadvantage, BASIC interpreters have been implemented on almost every microcomputer and are widely used for business, scientific, and personal computer applications. This wide acceptance is due mainly to the interactive nature of interpreters.

The PDOS BASIC interpreter combines compiler performance with the convenience of an interpreter in a unique approach to program developement. BASIC commands are parsed into executable tokens during program entry and not at execution time. Hence, program lines do not require needless, time consuming recursive parsing every time they are executed. The BASIC interpreter executes as fast as any threaded code compiler.

Program development time is greatly reduced due to the interactive nature of PDOS BASIC. A program can be interrupted, variables examined and changed, program lines altered and added, trace parameters set, and then execution continued. Most commands can be executed directly from the keyboard.

Program labels and multi-lined functions enhance the structured design and readability of a user program. Transfers can be made to meaningful labels rather than just to line numbers. Variable names can be of any length rather than the regular one or two character names found in other BASICs. Local variables within functions improve program integrity.

## FUNCTIONAL DESCRIPTION:

STANDARD DARTMOUTH BASIC COMMANDS. PDOS BASIC supports most commands commonly found in BASIC interpreters. LET variable assignment, FOR/NEXT loops, IF/THEN statements, GOTO/GOSUB/RETURN transfers, and READ/DATA program statements are standard. All standard operators (+, -, *, /, etc.) and system functions (LOAD, SAVE, RUN, NEW, etc.) are included.

BASIC ENHANCEMENTS. In addition to standard BASIC commands, PDOS BASIC allows multi-dimensioned arrays. Array sizes are not limited to 64K bytes. Variable names can be of any length. The ELSE statement complements THEN. Subroutine calls can be by name as well as by line number. A program can be listed according to token storage so that exact execution order can be verified. Bit and address functions give user programs control over variable storage and formats.

BASIC FUNCTIONS. PDOS BASIC functions are recursive and can be either single or multi-lined. Up to seven local arguments can be passed to the function and other variables can also be declared as local.

STRING OPERATIONS. For speed and convenience, strings are context orientated. Variable data can be interpreted as an integer, a floating point number, or a string of ASCII characters. Only the context in which a variable is used dictates how the data is to be treated. Command functions of string assignment, concatenation, deletion, insertion, length, search, replacement, extraction, and numeric conversion are included.

ASSEMBLY LANGUAGE SUPPORT. Assembly language support is an important feature found in PDOS BASIC. Assembly routines, either loaded from disk or generated from DATA statements, are executed from within BASIC variables. The subroutine linkage is well defined and parameter passing, using integers, is simple. The EXTERNAL command further simplifies the linkage process. New meaningful verbs can be added to the BASIC command list and external routines are called by BASIC whenever the verb is used.

PROGRAM DEBUGGING. A single step feature in PDOS BASIC allows a user program to be executed a single line at a time. All assignments can be displayed and all program transfers indicated. Additionally, selected variables can be tagged to display whenever altered. A program can be interrupted and continued after examining and even altering program lines and variables. A program line can be displayed for editing without having to either retype the line or enter a special edit mode.

INTER-TASK COMMUNICATION. BASIC program tasks can communicate with other tasks using events or mailboxes. A special MAIL array is global to all tasks and can be used for sending and recieving messages. Event commands allow BASIC programs to synchronize with other tasks. A GLOBAL command allows many BASIC programs to share the same variables.

REAL-TIME SUPPORT. Special BASIC commands have been added to suspend a program while waiting for a software or hardware event. Time and date parameters are available as well as delta time functions. Timeout events may be included to prevent system lockouts.

FILE MANAGEMENT. A full complement of file commands is supported by PDOS BASIC. These include open, read-only open, random open, and shared random open, define, delete, reset, rename, read, write, position, lock, unlock, load, and save. Also, a BASIC program can be saved in token form for extremely fast loading.

STANDALONE RUN-TIME SUPPORT. To generate a ROMable, standalone execution module, a debugged BASIC program can be linked to the PDOS run-time kernel, along with other tasks and support routines. A 32-bit floating point version of the BASIC module can be selected for a further reduction in execution time and memory size.

SYSTEM FUNCTIONS AND OPERATORS. Logical, arithmetic, and boolean operators are all available in PDOS BASIC. System functions allow various execution parameters to be examined and changed including input and output ports, memory limits, stack sizes, etc.

## Example Program for PDOS BASIC

```
LIST

100 PRINT : INPUT "DISTANCE=";X

110 INPUT "MUZZLE VELOCITY=";V

120 T=FNS[0,ATN 1]

130 IF T<0: GOTO 100

140 PRINT "ELEVATION IS";T*180/3.141592654;

150 PRINT " DEGREES"

160 PRINT X/(COS[T]*V);" SECONDS OF FLIGHT"

170 GOTO 100


500 DEFN FNA[A]=-9.8*X/(V*COS[A])+2*V*SIN[A]


600 FOR I=1 TO 20

620  II=(E1+E2)/2: FNS=II

630  IF FNA[II]*FNA[E1]<=0: E2=II: GOTO 670

640  IF FNA[II]*FNA[E2]>0

650   THEN PRINT "NO SOLUTION": FNS=-1: FNEND

660   ELSE E1=II

670 NEXT I

680 FNEND
```

Dear Customer,

While FORCE Computers has achieved a very high standard of quality in our products and documentation, we continually seek suggestions for new improvements. We would appreciate any feedback you care to offer.

Please use the attached "PRODUCT ERROR REPORT" form for your comments and return it to one of our FORCE Computers offices.

Sincerely,
FORCE Computers.

# P R O D U C T   E R R O R   R E P O R T

HARDWARE/SOFTWARE/SYSTEMS

PRODUCT : ____________________

SERIAL NO. : ____________________

DATE OF PURCHASE : ________________

ORIGINATOR : ____________________

COMPANY : ____________________

ADDRESS : ____________________

____________________

____________________

DATE : ____________________

TELEPHONE : ( ) EXT ______

CONTACT : ____________________

This box to be completed by FORCE

DATE : __________

PR# : __________

ACTION BY :

Engineering ( )
Marketing ( )
Production ( )

AFFECTED PRODUCT: ( ) SOFTWARE ( ) HARDWARE ( ) SYSTEM

AFFECTED DOCUMENTATION: ( ) SOFTWARE ( ) HARDWARE ( ) SYSTEM

ERROR DESCRIPTION : ____________________

SEND TO :

| | | |
|---|---|---|
| FORCE Computers Inc. | FORCE Computers GmbH | FORCE Computers FRANCE |
| Marketing | Marketing | Marketing |
| 727 University Avenue | Daimlerstrasse 9 | 11, rue Casteja |
| Los Gatos, CA 95030 | 8012 Ottobrunn/Munich | 92100 Boulogne |
| U.S.A. | West Germany | France |