CHAPTER 3

PDOS

The PDOS operating system is described here in detail. There are four main sections of PDOS; namely, the kernel, the file management module, the command line interpreter or monitor, and the floating point package.

3.1 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 kernel:

1. Multi-tasking, multi-user scheduling
2. System clock
3. Memory allocation
4. Task synchronization
5. Task suspension
6. Event processing
7. Character I/O including buffering
8. Support primitives

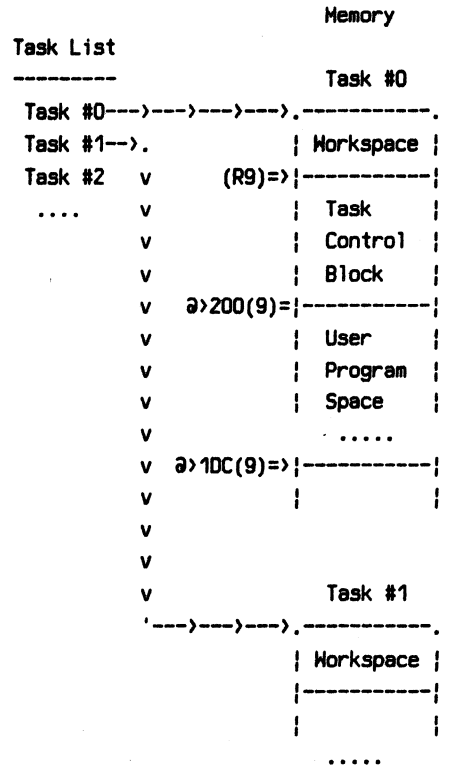
3.1.1 PDOS TASK

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 workspace, 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 is 20 bytes long and consists of a page/count, task number, task control block pointer, workspace pointer, program counter, status register, floating point accumulator, and error return. (A 102 system also includes 24 bytes of mapping information.)

The user workspace is the first 32 bytes of the task memory. All registers are available for use by a task if desired.

Immediately following the workspace is the task control block. This block of memory consists of three buffers, three additional workspaces, and parameters peculiar to the task. Register R9 of the user workspace points to the status block when the user program space is entered. The task parameters may be referenced by a user program but care must be taken that PDOS is not crashed! The task control block variables are displacements beyond register R9 and are defined in FIGURE 3.1.



(3.1.1 PDOS TASK continued)

The user program space begins immediately following the task control block. Relocatable 9900 object programs or BASIC tokens are loaded into this area for execution. Task memory is allocated in either 1k or 4k byte increments depending upon the type of system. The total task overhead is >220 or 542 bytes. This leaves >1E0 or 480 bytes available for a user program in a minimal 1k byte task.

Task overhead = 542 bytes

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:

4 task states:

1. Undefined
2. Ready
3. Running
4. Suspended

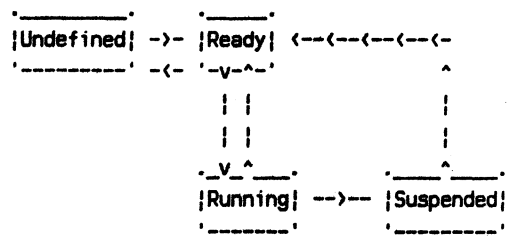
1. Undefined A task is in this state before it is loaded into the task list. It can be a block of code in a disk file or stored in memory.

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



3.1.2 MULTI-TASKING

16 independent time shared tasks

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

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

1) A time interval indicating how long the task executes before being swapped to the next task by the system clock - defined in clock tics.

1 tic = 1/125 second

2) The task memory requirement in 1k byte increments.

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

4) A task command.

Task defaults

Each of the above requirements defaults to a system parameter. For instance, default time slice is three tics (3 x 8 milliseconds = 24 milliseconds). Default memory allocation is 1k bytes and default console port is the phantom port.

Automatic task termination

If a task command is not specified, the new task reverts to the PDOS monitor. However, if no input is possible (ie. port 0 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).

Task entry in task list

A task entry in the task list queue is 20 to 44 bytes long and consists of a task number designation, parent task number, time interval, memory page number, task control block pointer, program counter, workspace pointer, status register, floating point accumulator, error register, and 12 mapping registers (PDOS 102). Swapping from one task to the next is done when the task interval timer decrements to zero or during an I/O call to PDOS. The task interval timer decrements by one every eight milliseconds.

(3.1.2 MULTI-TASKING continued)

Task memory allocation

Any task may spawn another task. Memory for the new task is allocated in 1k 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 amount of memory allocated to the new task.

Memory bit map

PDOS maintains a memory bit map to indicate which segments of memory are currently in use. Allocation and deallocation are in 1k 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 other tasks. Furthermore, when PDOS prompts for a new command, the memory bit map is checked for any available memory adjacent to the upper limit of the task. If more memory is available, it is allocated to the task and the upper limit of the task is extended. Thus, memory is automatically recovered by the parent task.

Memory automatically recovered

Multi-user system

"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 0 is referred to as the system task and cannot be terminated.

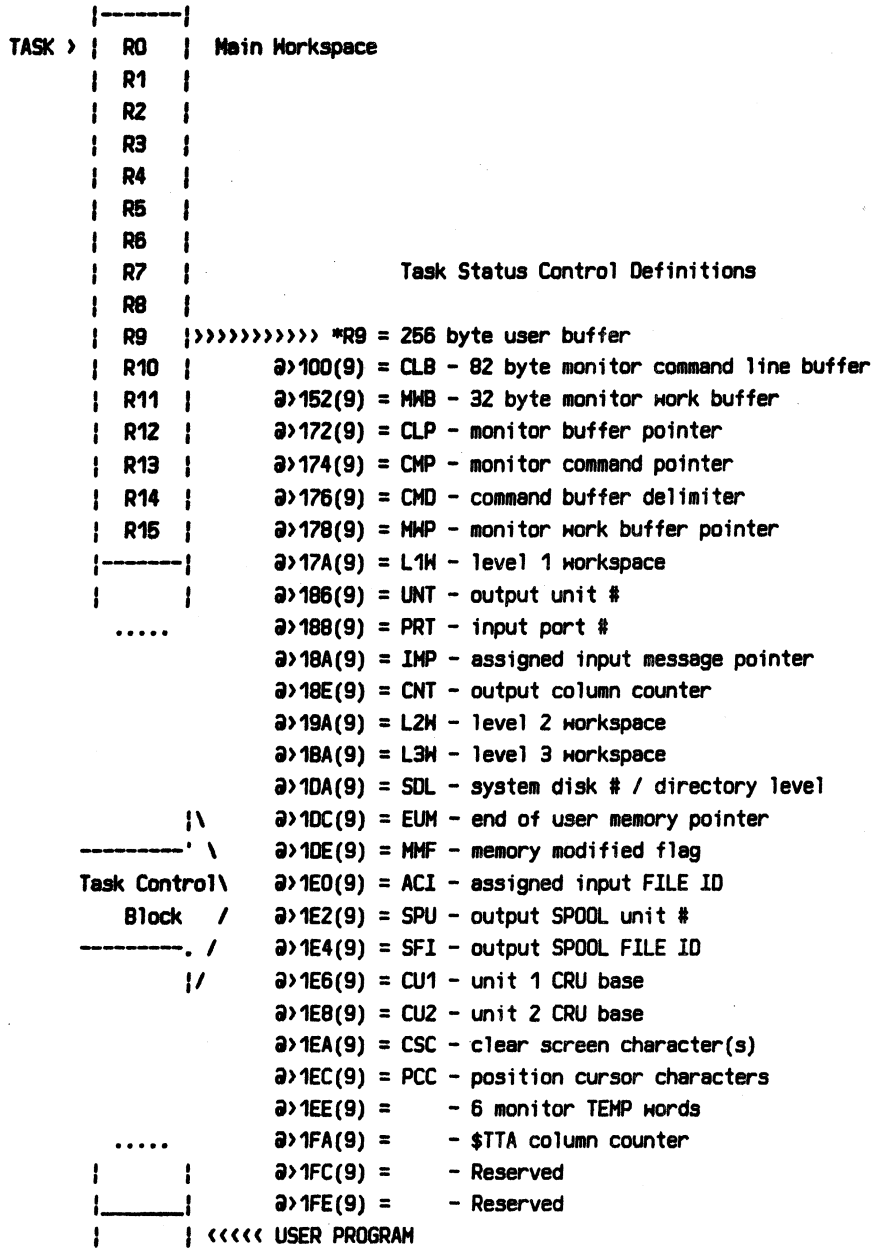


FIGURE 3.1 TASK CONTROL BLOCK

3.1.3 SYSTEM SERVICES

System services

System services are those functions that a task requires of the operating system while entered in the task list. These requirements range from timing and interrupt handling to task coordination and resource allocation.

Time keeping facilities

In addition to a variety of system tables, PDOS provides several time keeping capabilities. These include the current time of day and date. Also, a 32 bit counter can be used for various delta time functions.

Interrupts

Hardware interrupts are processed by the kernel or passed to user tasks. Tasks can be suspended pending the occurrence of an interrupt and then be rescheduled when the interrupt occurs. Interrupts such as the interval timer and character input or output are handled by the kernel itself.

Task coordination

Task coordination is an integral part of real-time applications since many functions are too large or complex for any single task. The PDOS kernel uses common or shared data areas, called mailboxes, along with a table of preassigned bit variables, called events, to synchronize tasks. A task can place a message in a mailbox and suspend itself on an event waiting for a reply. The destination task is signaled by the event, looks in the mailbox, responds through the mailbox, and resets the event signaling the reply.

System resources

System resources include the processor itself, system memory, and support peripherals. The PDOS kernel provides primitives to create and delete tasks from the task list. Memory is allocated and deallocated as required. Peripherals are generally a function of the file manager but are assigned and released via system events. Device drivers coordinate related I/O functions, interrupts, and error conditions. All of these functions are available to user tasks and thus tasks may spawn tasks and dynamically control their operating environment.

Support utilities

Other support utilities contained within the PDOS kernel include number conversion, command line decoding, date and time conversions, and message processing routines. Facilities are also provided for locking a task during critical code execution.

3.1.4 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.

Interrupt driver character type-ahead

Program control output

Inputs are through logical port numbers, whereas outputs are to physical CRU based UARTs (Universal Asynchronous Receiver/Transmitters). A logical input port is bound to a physical UART by the baud port commands and is uniquely assigned to a task. Many tasks may share the same output UART but must coordinate all outputs themselves.

Inputs through logical ports

Outputs through physical CRU bases

PDOS CHARACTER INPUT

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(9)), Assigned Console Input (ACI(9)), and input port number (PRT(9)).

Character inputs:

1. User memory
2. PDOS disk file
3. PDOS I/O device driver
4. System input port buffer

When a request is made by a task for a character and IMP(9) is nonzero, then a character is retrieved from the memory location pointed to by IMP(9). IMP(9) is incremented after each character. This continues until a null byte is encountered, at which time IMP(9) is set to zero.

```
IMP     EQU >18A        ;INPUT MESSAGE POINTER
        LI R1,CMMD      ;POINT TO COMMAND
        MOV R1,@IMP(9)  ;SET INPUT POINTER
        ....
CMMD    TEXT 'MESSAGE'
        BYTE 0
```

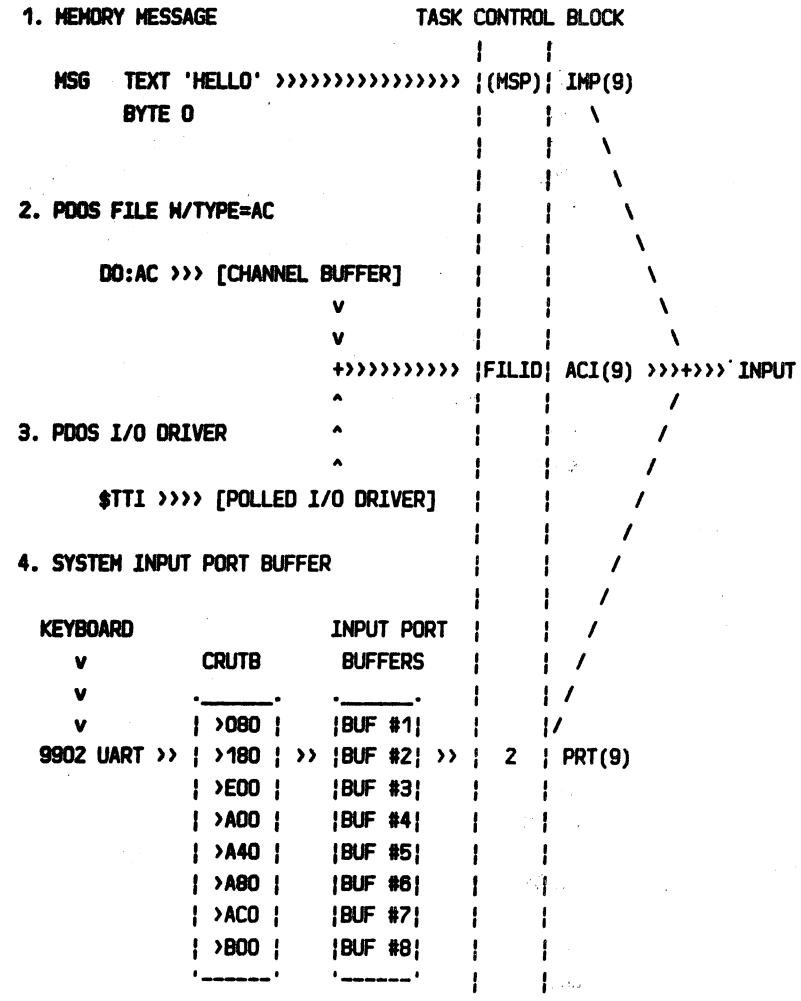
If IMP(9) is zero and ACI(9) is nonzero, then a request is made to the file manager to read one character from the file assigned to ACI(9). The character then comes from a disk file or an I/O device driver. This continues until an error occurs (such as an END-OF-FILE) at which time the file is closed and ACI(9) is cleared.

```
ACI     EQU >1E0        ;ASSIGNED CONSOLE INPUT
        LI R1,FILEN     ;POINT TO FILE NAME
        XSOP            ;OPEN FILE
          XERR
        MOV R1,@ACI(9)  ;SET CONSOLE INPUTS
        ....
FILEN   TEXT 'INDATA'
        BYTE 0
```

If both IMP(9) and ACI(9) are zero, then the logical input port buffer selected by PRT(9), is checked for a character. If the buffer is empty, then the task is automatically suspended until a character interrupt occurs.

PDOS character input flow is summarized by Figure 3.2.

```
PRT     EQU >188        ;INPUT PORT NUMBER
        LI R1,3         ;READ CHARACTERS FROM
        MOV R1,@PRT(9)  ;  PORT #3
        ....
```



NOTES: 1) CRUTB binds a physical 9902 UART to a logical port number.

2) 9902 UART baud rates and CRUTB entries are defined by the 'BP' and 'BAUD' commands (XBCP primitive).

3) XGCC gets characters from input port buffers only.

FIGURE 3.2 PDOS CHARACTER INPUTS

(3.1.4 PDOS CHARACTER I/O continued)

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(9)), spooling unit (SPU(9)), spooling file ID (SFI(9)), unit 1 CRU base (U1C(9)), and unit 2 CRU base (U2C(9)). 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(9). 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 a 9902 UART with CRU base U1C(9). Likewise, if the 2 bit is set in the masked output unit, then the character is output to the U2C(9) CRU based 9902 UART.

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 U1C(9) and U2C(9) variables within the task control block. Other unit bits are used for outputs to files and device drivers. Thus, if SPU(9)=4 and UNT(9)=7, then output would be directed to the file manager via SFI(9) and to two 9902 UARTS as specified by CRU bases in U1C(9) and U2C(9). (See Figure 3.3.)

```
@SPU(9) = 0000 0000 0000 0100
@UNT(9) = 0000 0000 0000 0111
                          ///
                         ///
        File @SFI(9)___///
        9902 @U2C(9)___//
        9902 @U1C(9)___/
```

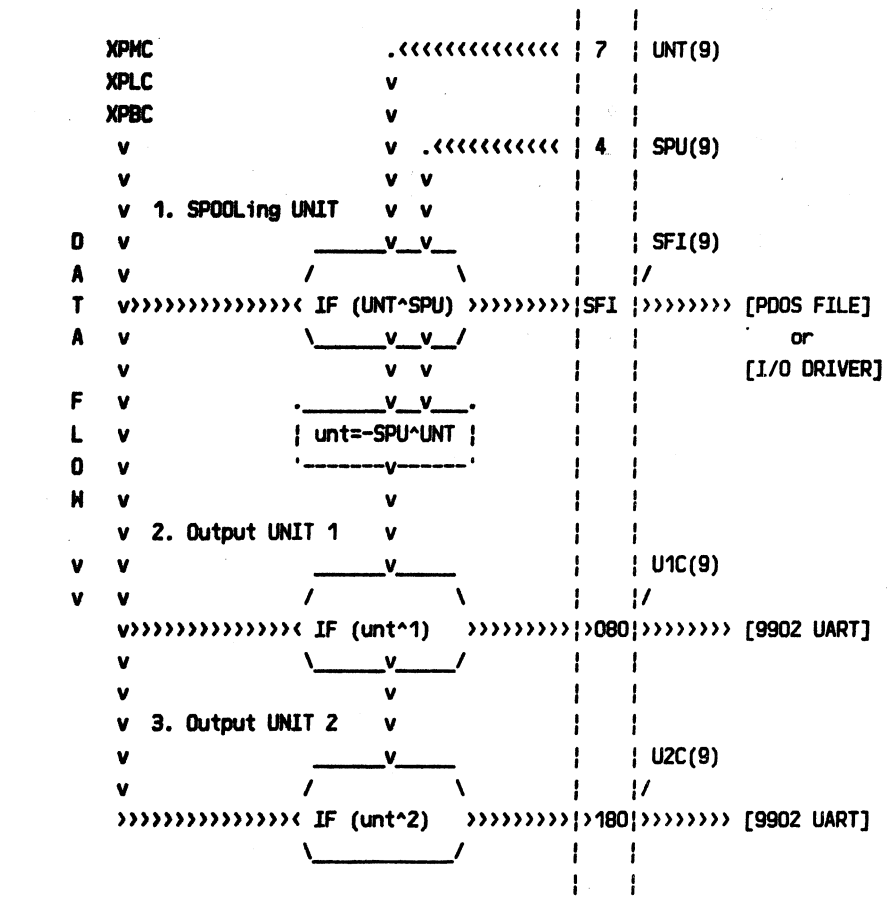
```
UNT     EQU >186        ;OUTPUT UNIT
SPU     EQU >1E2        ;OUTPUT SPOOLING UNIT
SFI     EQU >1E4        ;OUTPUT SPOOL FILE ID
U2C     EQU >1E6        ;UNIT 1 CRU BASE
U2C     EQU >1E8        ;UNIT 2 CRU BASE
*
        LI R1,FILEN     ;GET FILE NAME
        XSOP            ;OPEN FILE
          XERR
        MOV R1,@SFI(9)  ;SET SPOOL FILE ID
        CLR R2          ;CLEAR COUNTER
        MOV @C4,@SPU(9) ;SET SPOOL UNIT TO 4
*
LOOP    MOV R2,R1
        MOV R1,@UNT(9)  ;SELECT UNIT
        XCBM            ;CONVERT NUMBER
          DATA MES01
        XPLC            ;OUTPUT MESSAGE
        INC R2          ;INCREMENT R2
        CI R2,8         ;8 TIMES?
          JLT LOOP      ;N
        ....            ;Y

C4      DATA 4
FILEN   TEXT 'OFILE'    ;OUTPUT FILE NAME
        BYTE 0
MES01   TEXT 'OUTPUT MESSAGE #'
        BYTE 0
```

```
UNIT 1 =        OUTPUT MESSAGE #1
                OUTPUT MESSAGE #3
                OUTPUT MESSAGE #5
                OUTPUT MESSAGE #7

UNIT 2 =        OUTPUT MESSAGE #2
                OUTPUT MESSAGE #3
                OUTPUT MESSAGE #6
                OUTPUT MESSAGE #7

OFILE =         OUTPUT MESSAGE #4
                OUTPUT MESSAGE #5
                OUTPUT MESSAGE #6
                OUTPUT MESSAGE #7
```



Notes: UNIT 1 9902 = (-SPU ^ UNT) ^ 1
UNIT 2 9902 = (-SPU ^ UNT) ^ 1
PDOS FILE = (SPU ^ UNT)

FIGURE 3.3 PDOS CHARACTER OUTPUTS

3.1.5 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.

Events synchronize tasks

There are four types of event flags in PDOS: hardware, software, software resetting, and system events. System events are further divided into input, output, timing, driver, and system resource events. System events are predefined software resetting events that are set during PDOS initialization.

4 types of event flags:

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

1) 1-15 Events 1 through 15 are hardware events. They correspond to interrupt levels 1 through 15 of the TMS9900 CPU. When a task suspends itself pending a hardware event, the system TMS9901 mask is enabled allowing the interrupt to occur. When the interrupt does occur, the task list is searched for the suspended task. If the current task has not locked itself in the execution state, then the new task is awakened, swapped in, and immediately begins executing. Otherwise, the suspended task is set in the ready state and executes when scheduled. In either case, PDOS disables the interrupt in the system TMS9901, thus allowing the awakened task to acknowledge the interrupt. Only one task responds to any single hardware event.

1-15 = Hardware events

2) 16-63 Events 16 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.

16-63 = Software events

3) 64-94 Events 64 through 94 are like the normal software events except that PDOS resets the event whenever a task suspended on that event is rescheduled. Thus, one and only one task is rescheduled when the event occurs.

64-94 = Software resetting events

(3.1.5 EVENTS continued)

4) 95-103 Events 95 through 103 correspond to input ports 0 through 8. 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.

95-103 = Input port events

5) 104-111 Events 104 through 111 are used when doing interrupt driven character output (XIPL) and signal that a null character has been encountered and the output is completed. Thus a task could send a complete line to a terminal and either continue executing or suspend itself until the line is printed.

104-111 = Output complete events

6) 112-115 Events 112 through 115 are timing events and are set automatically by the PDOS clock module according to intervals set 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 525 seconds or 8.7 minutes. Events 113, 114, and 115 have a maximum interval of 65536 seconds or approximately 18.2 hours. A task suspended on one of these events is regularly scheduled on a tic or second boundary.

112 = 1/5 second event
113 = 1 second event
114 = 10 second event
115 = 20 second event

7) 116-127 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.

116 = $TTA active
117 = $LPT active
118-125 = To be assigned
126 = Error message disable
127 = System utility

3.1.6 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 maintains eight message buffers for queued message communications between task console terminals. More sophisticated methods require program arbitrators and message buffers as loaded by the ALOAD utility. A few methods are defined below.

Mailbox communication

MAIL array

MAIL[0] - MAIL[9]

The MAIL array is a permanent 60 byte memory buffer accessible by assembly language programs and PDOS BASIC as the singly dimensioned array MAIL[0] through MAIL[9]. The array is located at memory addresses >2204 through >223F. This array is never cleared even during PDOS initialization. (See 10.59 MAIL.)

COM array

COM[0] - COM[9]

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. (See 10.13 COM.)

Absolute data movement

MEM[adr]=data
MEMW[adr]=data
MEMP[adr,page]=data

Absolute memory locations are referenced by using the MEM functions. The MEM function moves byte data; MEMW moves words; and MEMP moves 6 byte BASIC variables. MEMP passes data between different memory pages or to a page external to the current task (102 system). (See 10.61 MEM through 10.63 MEMP.)

(3.2.6 TASK COMMUNICATION continued)

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 1 through 15 are defined as hardware events because they correspond to the 15 levels of interrupt of the TMS9900. Events 16 through 63 are for software communication flags. Events 64 through 127 automatically reset when a suspended task is rescheduled. Events 96 through 103 are input events; 104 through 111 are output events; 111 through 115 are timing events; and 116 through 127 are system events. (See 10.28 EVENT and 10.29 EVF.)

127 Event flags

EVENT 30

IF EVF[30]

Message buffers

PDOS maintains eight 50-byte message buffers for intertask communication. A message consists of up to 50 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.

16 50-byte buffers

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. (See 13.1 ALOAD and 13.20 FREE.)

Memory Mailbox

3.1.7 TASK SUSPENSION

Any task can be suspended pending a hardware or software event. Hardware events (1-15) correspond to the TMS9900 interrupt levels. Software events (16-127) are system memory bits global to all tasks. A suspended task does not receive any CPU cycles until the desired event 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 a minus event number being listed for the task time parameter.

When the event occurs, the task is rescheduled and resumes execution. If the event is a hardware interrupt (events 1 through 15), then the task is immediately rescheduled, overriding any current task. If the event is a software event (16 through 127), then the task begins execution during the normal swapping function of PDOS. (See 5.2.16 XSUI - SUSPEND UNTIL INTERRUPT and 10.106 WAIT.)

Task suspended pending event

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

 *0/0   0     3    >42A2 >441C >0654 >D40F ...
  1/0   0    -30    >4AA2 >4A82 >1040 >000F ...
  2/0   0    -5     >52A2 >5282 >292E >C40F ...
```

Hardware event response immediate

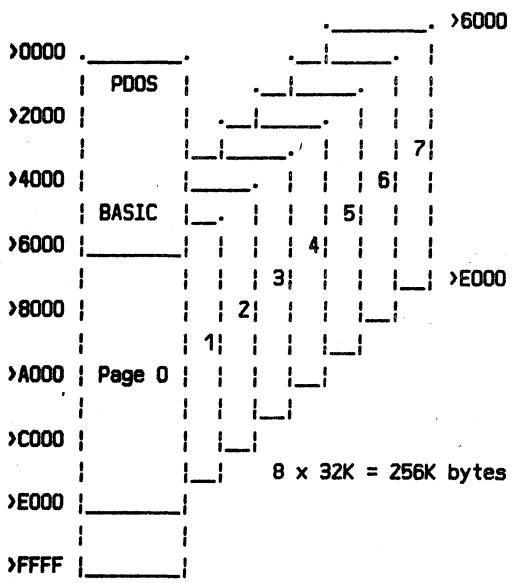
Software event response slower

3.1.8 MULTI-PAGING

Associated with each task is a 3 bit memory page number. The page number is output on the CRU bus at the beginning of each task time slice. This number is designed to select one of eight 32K byte memory pages and deselect all others. Thus, a system can handle up to eight memory planes or 256K bytes of user task memory.

Each memory plane has its own select logic. The memory addresses range from >6000 to >DFFF. The page select bits are at CRU base address >0980.

Intertask communication between different memory planes must be through the common memory plane from >0000 to >5FFF. The IMP utility installs a new memory page by setting the associated bits in the memory bit map. The MEMP function of PDOS BASIC is used to reference data in another page.



3.1.9 INTERRUPTS

PDOS supports user interrupt routines for levels 1, 2, and 9 through 15. Level 3 is reserved for the system clock. Levels 4 through 8 are dedicated to user 9902 terminal I/O. Uninitialized TMS9902 ports generate spurious interrupts. PDOS sets the CPU's interrupt mask to level 5 and enables interrupts 3 through 6 at the system 9901. This allows the system clock (level 3), system console (level 4), and aux port (level 5) to interrupt.

Before setting a new interrupt level with the INTERRUPT MASK command (IM), caution should be taken to ensure that all TMS9902's in the system have been reset and defined with the BAUD PORT (BP) command. Otherwise, the system hangs on spurious interrupts! The interrupt mask must always be greater than level 3 for system tasking and terminal access to work.

If a TMS9902 is installed on level 7 or 8, the corresponding mask bit must be enabled in the system 9901. This mask is located at memory location >0094 (>00B4 for 102) and is changed by the BFIX utility.

Levels 1-2 = High priority user interrupts
Level 3 = System clock
Levels 4-8 = 9902 console terminal I/O
Levels 9-15 = User interrupt routines

3.2 PDOS FILE MANAGEMENT

File management module

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 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.

Sequential, random, read only, and shared file access

3.2.1 PDOS FILE STORAGE

File, file directory

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.

Disk number

256 byte blocked data transfers

File directory entry

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 '$' preceding a file name 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.

Sector bit map

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 0) 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.

(3.2.1 PDOS FILE STORAGE continued)

PDOS file slots

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 outside 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 in the file slot status area current file information 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.

Sector buffer and status area

32 simultaneously OPENed files

Up to 32 files may be open at a time. Keeping all sector buffers resident would require prohibitive amounts of system memory. Therefore, only four 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.

4 active buffers

Most-recently-accessed resident buffer allocation

Frequent access = fast access

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.

Forward and backward linked sector file storage

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.

Sequential or random access

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.

(3.2.1 PDOS FILE STORAGE continued)

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.

```
Shared random, read only random access

Shared random access
```

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.

```
Read only random access
```

3.2.2 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 separated 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 0 to 255. The disk number ranges from 0 to 127.

```
FILE
A1234567:890;255/127
PROGRAM/3
FILE2;10
```

A file name beginning with a dollar sign is processed by PDOS as a system I/O device driver. Entry points are provided directly into the channel buffer for OPEN, CLOSE, READ, WRITE, and POSITION commands.

```
$TTO,$TTA,$LPT,$CRD
```

If the file name is preceded 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.

```
Auto define

.CF TEMP,#TEMP2/5
```

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 parsing a command line.

```
FILEN   TEXT 'FILE1/4'
        BYTE 0
```

3.2.3 DIRECTORY LEVELS

Each PDOS disk directory is soft partitioned into 256 directory levels. Each file resides on a specific level, which facilitates selected directory listings. You might put system commands on level 0, procedure files on level 1, object files on level 10, listing files on level 11, and source files on level 20. All files are global with respect to a disk directory and can be accessed without referencing the file level.

```
256 directory 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. File names are not unique to a level, hence the same file name cannot be used twice in any one disk directory.

```
.LV
LEVEL=1
.
```

3.2.4 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 into 256 byte records called sectors.

```
.SY 1
.SY
SYS DISK=1
.
```

The range of disk numbers is from 0 to 127. Several disk numbers may share the same secondary storage device. Each disk can have a maximum of 65280 sectors or 16,711,680 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.

3.2.5 FILE ATTRIBUTES

Associated with each file are file attributes. 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.

8 defined file types

The file type is used by the PDOS monitor in processing the 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 9900 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.

Relocatable object only

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 from the assigned file.

Batch processes

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

OB - 9900 tag object file. All assembly user defined commands are typed as object files. This directs the PDOS monitor to load the file into memory and execute the program.

Must be relocatable object

SY - System file. A 'SY' file is generated from an 'OB' file. TI9900 object is condensed into a smaller and faster loading format by the 'SYFILE' utility.

Generated from OB file

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 restore the tokens for the BASIC interpreter and begin execution.

SAVEB "FILE"

(3.2.4 FILE ATTRIBUTES continued)

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.

SAVE "FILE"

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.

UD - User Defined. A 'UD' file type has no significance to PDOS other than file classification.

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 delete 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.

Contiguous File

* - Delete protect. A file which has one asterisk as an attribute cannot be deleted from the disk until the attribute is changed.

Delete protect

** - 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.

Delete and write protect

3.2.6 TIME STAMPING

When PDOS is first initialized, the system prompts for a date and time. These values are then maintained by the system clock and are used for time stamping file updates, assembly listings, and other user defined functions.

```
PDOS/101 R2.4
ERII, COPYRIGHT 1982
DATE=MN,DY,YR 3,5,82
TIME=HR,MN,SC 12,01
```

When a file is first created or defined, the current date and time is stored with the disk directory entry. This time stamping appears in the 'DATE CREATED' section of a directory listing. From then on, the creation date and time are not changed.

```
Date created
```

When a file has been opened, altered, and then closed, the current date and time are written to the 'LAST UPDATE' section of the disk directory entry. The time stamp indicates when the file was last altered by any user.

```
Last update
```

3.2.7 PORTS, UNITS, AND DISKS

The terms ports, units, and disks are often confused and hence are explained again:

Ports — Ports are logical input channels and are referenced by numbers 0 through 8. Associated with each port is an interrupt driven input buffer. The BAUD PORT command binds a physical 9902 UART to a buffer.

Units — A unit is an output gating variable. Each bit of the variable directs character output to a different source. Bit 1 (LSB) is associated with U1C(9) CRU base. Likewise, bit 2 is associated with U2C(9) CRU base. The 'SU' and 'SPOOL' commands bind the other bits to the PDOS file structure.

Disks — A disk is a logical reference to a secondary storage device. Disk numbers range from 0 to 127. Several disk numbers may reference the same physical device. The boot EPROMs decipher what the disk number means.

3.3 PDOS MONITOR

The PDOS monitor is a resident program which handles the most common PDOS commands. After getting a command line, the monitor calls the command line interpreter to parse the line for commands and parameters. A command line is delimited by a <carriage return>. If a command line is not complete, your task is suspended pending character inputs.

The PDOS monitor prompts with a bell followed by a period. These characters are altered by the BFIX utility. A command line can be up to 78 characters. The escape <esc> or control C (^C) keys cancel the entire input line. The rubout <rub> key erases the last entered character from both the input line and the character buffer.

80 character command buffer

A bell signals one of the following: 1) the monitor is ready to accept a line, 2) a rubout is entered and the buffer is empty, 3) too many characters are entered, or 4) the number of characters equals the internal message buffer size. The latter indicates the maximum command string length that can be passed to another task.

```
Bell => Buffer ready
        Buffer underflow
        Buffer overflow
        Message buffer equivalent
```

Numeric parameters are entered as signed decimal, hex, or binary numbers. All numbers are converted to 2's complement 16-bit integers and range from -32768 to 32767 (hex >8000 to >7FFF). Hex numbers are preceded by a right angle bracket (>) and binary numbers by the percent sign (%). (Note: Numbers are not checked for overflow. Hence, 65535 is equivalent to -1.)

```
.IM >0F
.CONVERT %1100101,10,>FFE2
```

You enter more than one PDOS command on a line by separating the commands with a period. Command parameters immediately follow the command name and are separated by commas or spaces. Nested parentheses are used to enclose parameters within parameters. When multiple commands appear on the same line, the remainder of the command line is echoed by the monitor as each command is executed.

```
.LS.SY 1.LS /1.LV

.CT (CT (ASM PRGM:SR,,LIST,ERR),10,,2),12,,2

.SP.LV.SY
FREE=180
USED=190/200
.LV.SY
LEVEL=1
.SY
SYS DISK=0
.
```

3.4 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. Commands include the following:

1. Addition
2. Subtraction
3. Multiplication
4. Division
5. Load accumulator
6. Store accumulator
7. Scale
8. Clear
9. Float
10. Normalize
11. Negate
12. Absolute value
13. Multiplicative inverse
14. Load clock tics
15. Load error register
16. Return accumulator status

Floating point operations are called with XOPs 0 through 8. The floating point accumulator is saved in the task list after each swap operation.