TOPS–10
Monitor Calls Manual
Volume 1

AA–0974F–TB

April 1986

This manual describes the functions that the monitor performs to service monitor calls from assembly language programs. The *TOPS–10 Monitor Calls Manual* is divided into two volumes: Volume 1 covers the facilities and functions of the monitor; Volume 2 describes the monitor calls, calling sequences, and GETTAB tables.

This manual supersedes the previous manual of the same name, order number AA–0974E–TB.

OPERATING SYSTEM:	TOPS–10 V7.03
SOFTWARE:	GALAXY V5.1

Software and manuals should be ordered by title and order number. In the United States, send orders to the nearest distribution center. Outside the United States, orders should be directed to the nearest DIGITAL Field Sales Office or representative.

Northeast/Mid–Atlantic Region

Digital Equipment Corporation
PO Box CS2008
Nashua, New Hampshire 03061
Telephone:(603)884–6660

Central Region

Digital Equipment Corporation
Accessories and Supplies Center
1050 East Remington Road
Schaumburg, Illinois 60195
Telephone:(312)640–5612

Western Region

Digital Equipment Corporation
Accessories and Supplies Center
632 Caribbean Drive
Sunnyvale, California 94086
Telephone:(408)734–4915

digital equipment corporation • marlboro, massachusetts

First Printing, January 1971
Revised, January 1972
Revised, March 1973
Revised, May 1974
Revised, November 1975
Revised, May 1977
Revised, January 1978
Revised, August 1980
Revised, February 1984
Revised, April 1986

The postpaid READER'S COMMENTS form on the last page of this document requests the user's critical evaluation to assist us in preparing future documentation.

CONTENTS

FIGURES

TABLES

PREFACE

The TOPS-10 Monitor Calls Manual is a complete reference set for using the TOPS-10 monitor calls. The set consists of two volumes:

Volume 1 contains descriptions of the facilities available to the assembly language programmer through the use of calls to the monitor. It details the requirements for performing various types of I/O, computation, and information processing using monitor-defined symbols and data. Specific monitor facilities, such as inter-process communication, programmed interrupts, and device I/O are each described in relation to the monitor calls needed to use those facilities.

Volume 2 lists the monitor calls themselves, in alphabetical order, including coding sequences for calling the monitor and for reading data returned by the monitor. The data may be returned on a successful completion of the call, or data on the error will be returned when an error occurs in the attempt to execute the monitor call. Volume 2 also contains a list of the GETTAB tables, which contain data about the monitor and user jobs. The data stored in these tables is extensive and yet easily accessible through the GETTAB monitor call. Finally, Volume 2 contains a glossary of the terms used in both volumes, a detailed description of the format of an executable file, and a description of the File Daemon program, which provides user-definable file security measures.

Before you attempt to use the TOPS-10 Monitor Calls Manual, you should have a basic familiarity with the structure, paging mechanisms, and hardware of the DECsystem-10. The TOPS-10 Monitor Calls Manual does not attempt to describe the monitor on a general level. You should also be familiar with the MACRO programming language before you read this manual. Specifically, it is recommended that you become familiar with the following manuals and topics before attempting to use the TOPS-10 Monitor Calls Manual:

- The DECsystem-10 MACRO Assembler Reference Manual is very important to an understanding of the methods for programming in assembly language on TOPS-10. The TOPS-10 monitor calls are written to facilitate the task of programming in MACRO, but the bulk of assembly language programming involves the operations described in the MACRO Reference Manual.

- Restrictions and capabilities of higher-level programming languages on TOPS-10 are not described in the TOPS-10 Monitor Calls Manual. If you are programming in a higher-level language (FORTRAN, COBOL, and so forth), you should obtain a copy of the programming language's specific reference manual written for TOPS-10.

- Facilities and capabilities of software products that run on TOPS-10 but are distributed as a separate product are not included in the TOPS-10 Monitor Calls Manual. Many references to such products (DECnet-10, IBM communication, and so forth) are made in the TOPS-10 Monitor Calls Manual, but you need the appropriate product-specific documentation to use the monitor facilities provided for these products.

The TOPS-10 Monitor Calls Manual is divided into two volumes only because there is a great amount of information that must be included in the manual. Therefore, neither volume can be used without the other.

Volume 1 describes the facilities your program can access through requests to the monitor; and contains specific references to calls, and calling functions; but only in Volume 2 can you find the detailed lists of functions available through each call, the specific kinds of data available after the execution of each call, and the restrictions and requirements for each.

Volume 2 contains detailed documentation of each monitor call, flag, argument block, and returned information, with programming requirements for each call, in a manner similar to the monitor source file UUOSYM.MAC. However, this information is useless without the knowledge available in Volume 1: the order in which calls must be made to the monitor, methods for handling errors, and the types of information you can use to make your programs interact smoothly with the monitor.

The TOPS-10 Monitor Calls Manual is intended to be the primary source of reference information for the interface between user programs and the monitor. If you find any errors in the manual, or have difficulty using the manual for any reason, please detail the problem on the Reader's Comment Card provided at the end of each volume and mail it to Digital Equipment Corporation. This important form of feedback is vital to the accuracy and usefulness of the documentation, and complete information about the problem would be greatly appreciated.

CHAPTER 1

INTRODUCTION TO MONITOR CALLS

A program written in MACRO-1Ø assembly language has four types of statements:

- Pseudo-op statements, such as BLOCK, TITLE, and RADIX, are instructions to the MACRO assembler and do not generate code.

- Direct-assignment statements, such as T1=1 and P=17, resolve definitions of symbols. Pseudo-ops and direct-assignment statements are described in the DECsystem-1Ø MACRO Assembler Reference Manual.

- Machine instructions, such as ADD, MOVEM, and JRST, are direct hardware instructions. Machine instructions and their symbols are discussed in the DECsystem-1Ø/2Ø Processor Reference Manual.

- Monitor calls, such as INPUT, CLOSE, and GETSTS, are directions to the monitor to perform special services for the program. Monitor calls, also called UUOs (Unimplemented User Operations), are described in this manual.

An operation code (opcode) and a symbol representing the name of the monitor call designates each monitor call. Use operation codes (opcodes) to direct the TOPS-1Ø monitor to perform I/O and other services for your program. Opcodes can be divided into the following groups:

- Opcode Ø always returns your job to monitor command level, because opcode Ø is an illegal UUO. The monitor displays the following error message, followed by the monitor prompt.

  ?Illegal UUO at user PC addr

- Opcodes 1 through 37 cause the hardware to store the instruction code and the effective address in location 4Ø, and to execute the instruction at location 41 in the user's address space. The original contents of location 4Ø are lost. This trap allows your program to gain control when using these opcodes. These opcodes can be defined by the system programmer as Local UUOs (LUUOs). If your program executes one of these opcodes accidentally, the monitor displays the message:

  ?HALT at user PC addr

  The monitor displays this message because relative location 41 contains a HALT instruction, unless the contents of location 41 were changed inadvertently. The LINK program provides the HALT instruction. To set or read trap

instructions for an LUUO in a non-zero section, you must use the UTRP. monitor call. Refer to Volume 2 for a description of the UTRP. UUO.

- Opcodes 40 through 100 are the opcodes for monitor calls. The TOPS-10 monitor defines these opcodes. They are called Monitor UUOs (MUUOs). This manual describes all monitor calls for TOPS-10. A monitor call is stored at location 424, the new PC is loaded from location 436 of the user's process table, and the processor operates in executive mode. The monitor interprets the opcode; then the monitor performs I/O and other control functions for your program.

- Opcodes 101 and 104 cause the monitor to stop your program and display the following message:

  ?Illegal instruction at user PC addr

- Opcodes 102, 103, 106, and 107 are legal only on the KL processor. On any other type of DECsystem-10 processor, the monitor will stop the program and display the following message if it receives one of these opcodes:

  ?KL10 only instruction at user PC addr

1.1 MONITOR CALL SYMBOLS

Opcodes 40 through 100 provide the basic set of monitor calls. (The opcodes and names of these calls are listed in Chapter 22, Volume 2.) Three of these calls (CALLI, MTAPE, and TTCALL) offer extended calls by using values in addition to the opcode value. (CALLIs and MTAPEs use the address field; TTCALL uses the accumulator field.) Each extended call has a symbol that defines its opcode and its value. These symbols and their full values are also listed in Chapter 22, Volume 2.

Most monitor calls accept arguments, return values, or both. Almost all these arguments and values have symbols defined in the monitor symbol file for UUOs, UUOSYM.MAC. Error codes, flag bits, and interrupt conditions all have symbols defined in UUOSYM.MAC. The file MACTEN.MAC contains definitions relating to the hardware, such as the PC flag bits. The file JOBDAT.MAC defines the job data locations. User programs should be written to reference all system values symbolically by these three universal files. That is, a SEARCH statement should appear near the beginning of the program to search UUOSYM, MACTEN, or JOBDAT. Refer to the TOPS-10 MACRO Assembler Reference Manual for a description of universal files and the SEARCH pseudo-op.

Some of the symbols represent codes that tell the monitor what is being specified; others are masks that you can use to isolate or test a returned value. For example, the symbol IO.ERR is defined as follows:

```
IO.ERR==17B21
```

This defines a 4-bit field in the I/O status word, allowing you to logically AND the returned file status word with the value IO.ERR to mask out all other bits of the word.

1.2 PROCESSING MODES

The DECsystem-10 hardware defines three processing modes: user mode, executive mode, and user I/O mode. Normally, programs run in user mode, which allows the processor to protect and map data in core successfully. The processor will switch from user mode to executive mode when a monitor call is issued. The monitor controls execution from that point until the monitor call finishes. User I/O mode, a privileged alternative to user mode, provides the program with direct access to I/O devices. This allows real-time device drivers to run under TOPS-10 in user mode.

1.2.1 User Mode

The majority of user programs execute in user mode. For a user-mode program, the processor:

- Performs automatic memory protection and mapping.
- Passes control to the monitor if a monitor call or an illegal instruction (including HALT) executes. In this case, the processor enters executive mode.

The hardware traps to location 40 in the job data area if an opcode less than 40 (but not 0) executes (see Chapter 3).

User mode requires an assigned area of memory. Illegal user mode instructions are:

- I/O instructions (opcodes 700 through 777) except those giving a device code greater than 734. KS10 I/O instructions are all illegal.
- All unimplemented opcodes (those not defined as machine instructions or monitor calls).
- Any JRST instruction with an accumulator greater than 2, except JRST 5 (XJRSTF) and JRST 15 (XJRST).

If your program executes an illegal instruction, one of the following messages prints (and your program stops):

```
?HALT at user PC addr
?Illegal instruction at user PC addr
```

where addr is the memory location of the illegal instruction.

For the HALT message above, you can continue the execution of your program at the target address of the JRST 4,(HALT) by giving the CONTINUE or CCONTINUE monitor command (see the TOPS-10 Operating System Commands Manual). For the "illegal instruction" message, you must correct your program and begin execution again.

1.2.2 Executive Mode

A user program switches to executive mode to perform a monitor call when it encounters an illegal instruction, or on a HALT instruction. The monitor always executes in executive mode, giving it special memory protection and mapping.

1.2.3 User I/O Mode

A program executes in user I/O mode if bits PC.USR and PC.UIO (bits 5 and 6) in the program counter (PC) word are set. These bits are defined in the MACTEN.MAC symbol file. I/O mode is the same as user mode, except that the hardware allows any opcodes or instructions to be executed except a HALT (JRST 4).

User I/O mode provides some protection against partially debugged monitor routines, and allows device service routines to be executed as user jobs. Realtime programs execute in user I/O mode, allowing them to gain direct control of devices. Refer to the DECsystem-10/20 Processor Reference Manual for information about processing modes, or to Chapter 9 of this manual for information about programming real-time devices.

To execute in user I/O mode, your job must have the JP.TRP (trap) or JP.RTT (realtime) privilege set, and you must successfully execute one of the realtime monitor calls TRPSET or RTTRP. When your program issues a RESET monitor call or clears it with a JRSTF, user I/O mode ends.

CHAPTER 2

MEMORY

Two distinct conventions allow you to reference DECsystem-10 memory: physical memory addressing and virtual addressing. Physical memory defines the physical limits of the CPU addressing space, and a physical address refers to an exact physical location within the memory unit of the processor. However, TOPS-10 is a timesharing system, characterized by the capability of serving multiple user programs simultaneously. A single program can be loaded into several different parts of memory that need not be contiguous. Therefore, user programs do not normally reference actual physical locations in memory.

Programs reference a self-contained set of "virtual" addresses. While the program is running, the virtual addresses are translated into physical locations that can be referenced by the hardware.

TOPS-10 provides each user with 512 pages of addressable virtual memory on the KS processor, and 16384 pages of addressable virtual memory on the KL processor. Although the conventional usage of the term "core" on TOPS-10 often is used to mean any type of memory, throughout this chapter, references to "memory" are to virtual memory and references to "core" are to physical memory.

The processor protects the memory needed for monitor-related functions, and the memory assigned to each job. It manages the assignment of memory space to each user, and controls the swapping of jobs into and out of core.

2.1 MEMORY ALLOCATION

Core memory is a physical and therefore finite space measured in 36-bit words, 512-word pages, or 1024-word K. Virtual memory uses these three measurements, and also 512-page sections. There are 32 (decimal) sections on the KL. They are referred to as Sections 0-37 (octal). A user's maximum virtual address space is 512P (pages) on the KS, and 16384P on the KL.

To accommodate all the users in the timesharing environment, TOPS-10 removes each job from core memory and places it in a special disk area temporarily to make room for another job. This function is called "swapping." You can prevent a program or program segment from being swapped out by using the LOCK monitor call to lock your job in memory. (This requires that the LOCK privilege be set in the privilege word in GETTAB table .GTPRV.) To use realtime devices, you need to LOCK your job in memory.

A user job need not be entirely in memory or on disk all at once. Your program can explicitly transfer individual pages of its virtual address space from core memory into and out of the working set. The working set is the collection of pages in core that are immediately accessible to a job, and those in core with the access-allowed bit off. Alternatively, if your program exceeds the current physical page limit for your job, it will be subject to paging by the monitor. Memory limits and paging are discussed later in this chapter. The TOPS-10 monitor itself always remains in memory.

On the KS, if you need to exceed the virtual memory limit of 256K, you must use an overlay structure. On the KL processor, if your program requires more than the virtual memory limit of 16384P, or you do not want to use extended addressing for a program greater than one section, you can construct your program using an overlay structure. Using overlays, your program can restrict its current virtual space to only a portion of the entire amount of references it may require. For information about using overlays, refer to the TOPS-10 Link Reference Manual.

The monitor enforces the following limits on memory space:

Symbol	Application
GPPL	(Global Physical Page Limit) is the maximum amount of core available to any user.
GVPL	(Global Virtual Page Limit) is the maximum amount of virtual memory available to any user. GVPL is 512P on the KS and 16384P on the KL. You may change this limit using the privileged SETUUO function .STMVM.
MPPL	(Maximum Physical Page Limit) is the maximum amount of core available to a given user.
MVPL	(Maximum Virtual Page Limit) is the maximum amount of virtual memory available to a given user.
CPPL	(Current Physical Page Limit) is the current amount of core that is available to the user, a value that must not exceed the user's MPPL.
CVPL	(Current Virtual Page Limit) is the current amount of virtual memory available to the user, a value that must not exceed the user's MVPL.

The monitor also maintains values to measure the amount of space currently being used and currently available to your job. Those are:

CPPC	(Current Physical Page Count) is the number of physical pages in core that are being used by your job.
CVPC	(Current Virtual Page Count) refers to the number of virtual pages that are being used by your job.

The monitor itself has a virtual address space greater than 256K because it uses KL-paging, the default paging method for monitors on KL and KS systems. Under KL-paging, multiple sets of 256K words each can be used by the monitor. Each set is called a "section." The KL processor supports up to 32 sections. Most of the monitor executes in Section 0. KL-paging allows the monitor to refer to code and data in non-zero sections. Refer to the Processor Reference Manual for more information about KL-paging.

The user core allocation cannot exceed MPPL, but you can adjust your program's limits from Ø to MPPL by using various functions of the SETUUO. You can set the size of CPPL to limit the physical size of your job. The sign bit of CPPL is symbolized as ST.VSG. If this bit is off, the limit is interpreted as a guideline by the page fault handler; if the bit is on, the value is interpreted as a strict limit. Use the CPPL word to control paging for your job. If your job exceeds CPPL, the monitor will initiate paging for your job by using the virtual memory software.

2.2 USER-MODE EXTENDED ADDRESSING

User-mode extended addressing gives you access to all 32 sections of virtual memory on the KL processor. 30-bit addressing allows you to reference any address in the 32 sections. Use extended addressing in situations where your program or data requires more than one section of virtual memory.

Your program may reference any section from any non-zero section. However, you may not reference a non-zero section from Section Ø, unless the section has been mapped together with Section Ø. Generally, however, if your program requires inter-section references, do not execute it in Section Ø.

When using user-mode extended addressing, make certain that the UUO argument list does not cross a section boundary. If it does, the argument list will wrap around to the beginning of the same section, overwriting the previous contents of those addresses, instead of continuing into the next section. To help you identify section and address within the section more easily, use the following format when writing the address:

```
section,,address
```

There are four ways to run your program in a section other than Section Ø. They are:

- Using the XJRST or XJRSTF instruction to place your program in an extended section. See the TOPS-1Ø/2Ø Processor Reference Manual for information about these instructions. The addresses in the program become thirty bits in the form: section,,18-bit addr.

- Placing a section number before a page number in any UUO that accepts page numbers as arguments. These UUOs include PAGE. and IPCFR..

- Supplying a 3Ø-bit argument to a UUO which allows extended addressing. UUOs that accept thirty-bit addresses include CTX., NSP., and ETHNT.. Note that the value in the AC is always interpreted as a global address for these UUOs, regardless of the section in which the UUO is executed.

- Formatting the core argument word to the GETSEG, MERGE, RUN, or SAVE UUO to include a section number.

2.3 USER MEMORY

You can exert considerable control over the functions of the monitor as it services your program and memory space. However, it is first necessary for you to understand the interaction of the monitor and your program. Before the monitor can run a program, it must be compiled or assembled, and loaded into core. This is discussed in more detail in Chapter 3.

The LINK program loads compiled programs (.REL files) into memory. If you save the core image of the loaded program, it is written to disk as an executable (.EXE) file, and need not be processed again by LINK. You can place an .EXE file into core (ready for execution) by using the GET or RUN monitor command. (See Chapter 3.)

2.4 CONTROLLING PROGRAM SEGMENTS

Programs that are loaded into memory can be divided into high and low segments. Every executable program has a low segment (lowseg); a program can also have a high segment (hiseg). A low segment is private. A program can always modify its own low segment.

A program might also have a high segment. By default, the high segment starts at virtual location 400000, and is either private or sharable. A sharable high segment can be used by more than one job. (For example, programs written in a high-level language, such as FORTRAN, use a sharable high segment. FORTRAN uses FOROTS.) By default, the monitor write-protects high segments so that no users can modify them. However, the owner of the file can clear the write-protect bit, allowing the program to modify the hiseg.

As an example of sharable high segments, suppose several users are executing FORTRAN programs. Each user has a low segment, containing most of his FORTRAN program. However, all FORTRAN users share the same high segment that contains the FORTRAN object-time system FOROTS. This conserves memory space used by programs that will not be modified.

The following sections describe the monitor calls that allow you to control the allocation, accessibility, and contents of program segments.

2.4.1 Adjusting the Size of Segments

You can change the core allocation of either or both of the low and high segments of your program dynamically by using the CORE monitor call. You cannot change the core allocation of segments or programs that are locked in core. You can use CORE to eliminate the high segment of your program, thus allowing you to create a new high segment. You should use separate CORE calls to change the sizes of the low and high segments to ensure that your program does not exceed its core limits. Memory that is allocated by CORE will be cleared before it is made available, to ensure privacy of user data. A CORE UUO function that does not alter the size of the program, or which decreases the program size, will clear any non-contiguous pages.

2.4.2 Merging Low Segments

You can merge portions of an .EXE file with the low segment of the program that is currently in memory by using the MERGE. UUO.

2.4.3 Writing Into the High Segment

You can set or clear the high segment write-protect bit for your program's hiseg by using the SETUWP monitor call. After the bit is cleared, you can modify your job's high segment.

You can replace or create a high segment for your job using the GETSEG call to read a hiseg from an .EXE file or the REMAP call to convert a contiguous portion of the low segment into the high segment.

The GETSEG monitor call replaces the current program's high segment with a new high segment from another .EXE file. The low segment of the .EXE file (if any) is not changed. This high-segment replacement is useful for sharing data, overlays, and runtime routines. The GETSEG monitor call accomplishes this in a manner similar to that used by the RUN monitor call (refer to Section 2.4.1.).

You can create a high segment from already-existing memory in your program's low segment by using the REMAP monitor call. Use REMAP to specify the virtual address where you want the hiseg to start. This allows you to define a contiguous portion of the low segment as the high segment. The specified portion of the low segment is removed and placed at the address in the REMAP call, which is the new hiseg origin address.

2.4.4 Testing for a Sharable High Segment

The bit SN%SHR in GETTAB table .GTSGN is set for each job that has a sharable high segment. The following code sequence shows how to use this bit to determine whether your own job's high segment is sharable. Note that -1 is used here for the job number to refer to your own job.

```
MOVE        AC1,[XWD -1,.GTSGN] ;Set up GETTAB
GETTAB      AC1,                ;Get hiseg parameters
  JRST      ERROR               ;GETTAB failed
TLNN        AC1,(SN%SHR)        ;Is it sharable?
JRST        NOTSHR              ;No
JRST        SHAR                ;Yes
```

2.4.5 Finding the Origin of a High Segment

The usual origin for a program high segment is location 400000; however, the origin can be at a different location. If you need to find the high segment origin for your program (for example, to access the vestigial job data area), you can get the address from the GETTAB table .GTUPM.

The following example shows how to obtain the high segment origin. Note that -2 is used here for the segment number to refer to the program's own high segment.

```
MOVE        AC,[XWD -2,.GTUPM]  ;Set up GETTAB
GETTAB      AC,                 ;Get origin
  JRST      NOHIGH              ;GETTAB failed
HLRZ        AC,AC               ;Set up origin
JUMPE       AC,NOHIGH           ;If Ø, no high segment
MOVEM       AC,HIORGN           ;Store hiseg origin
                                ;Code for no hiseg
```

2.4.6 Modifying a High Segment and Meddling

The monitor sets the write-protect bit for each new high segment. You can clear this bit using the SETUWP monitor call and you can increase or decrease the shared core using the CORE monitor call.

These calls are legal from either the low or high segment if the following is true:

- You have write privileges for the file from which the high segment was loaded.
- The segment has not been "meddled." Meddling occurs when a program attempts to modify a sharable hiseg, because such modifications might interfere with the other jobs using the hiseg.

A sharable high segment is considered to be meddled if any of the following is true:

- The program was started with a RUN monitor call that specified an offset start address other than Ø or 1, or with a START or CSTART command that supplied an address.
- The D (deposit) monitor command was used.
- The high segment was obtained with a GETSEG monitor call.

It is not considered meddling if you perform any of the above commands or calls with a nonsharable high segment.

If you have privileges to write to the file from which the sharable high segment came, you can use the D monitor command and the SETUWP and CORE calls for that high segment without meddling. You can write a program that accesses a sharable high segment using the GETSEG monitor call, and then turn off the write-protect bit with the SETUWP call.

When a sharable program has been meddled, the monitor sets the meddle bit for the meddling user. If the user attempts to clear the write-protect bit with a SETUWP monitor call, or to change the high-segment core assignment with a CORE monitor call (except to remove it completely), the monitor takes the error return. If the meddling user attempts to modify the high segment with a D monitor command, the monitor prints the following message:

```
?Out of bounds
```

Whenever the hiseg has been meddled, the monitor resets the user-mode write-protect bit to protect the high segment.

If you are suitably privileged, you can supersede a sharable high segment. When you use a CLOSE, OUTPUT, or RENAME monitor call, or some of the functions of the FILOP. UUO, on the file from which the sharable high segment was initialized, the monitor zeros the word containing the segment name in the GETTAB table .GTPRG. Jobs currently using that high segment can continue to do so, but after those jobs are completed (that is, when the segment becomes dormant), the monitor deletes that segment. Meanwhile, new users are able to share the new (superseding) segment.

If your program modifies a sharable hiseg, you are responsible for coordinating the changes with other users of the hiseg. Remember that your program can be interrupted within the execution of instructions that require multiple services, such as monitor calls, and a concurrent user program can be started at that time. On a multiple-CPU system, your program can be running simultaneously with another user program. Generally, when modifying a shared hiseg, you should refer to shared data with non-interruptable instructions, or under the protection of an interlock such as that provided by the ENQ/DEQ facility (see Chapter 8).

Because a sharable hiseg can exist on the swapping space after all users have released it, your program must be prepared to handle inconsistencies in data or "stale" interlocks left by previous user programs that ended prematurely.

2.5 RUNNING A PROGRAM

The RUN monitor call transfers control to another program by replacing both segments of the current program with the specified program, and starting execution of that program at its normal start address or at a specified offset from the start address. The new program completely replaces the calling program, so that there is no return to the calling program (unless you RUN it later).

When the RUN monitor call is executed, the monitor clears all of your job's memory. Your programs, however, should not assume that this action will occur. You must initialize memory to the desired values to allow your programs to be restarted by the CTRL/C and START sequence.

If you want to call a program from a system library, your program should call it by using device SYS, and a zero project-programmer number. The extension you specify for these programs should also be zero.

The RUN UUO executes a RESET monitor call, which (among other functions) releases all user I/O channels.

The RUN call uses the following information in its ac:

```
start-addr-offset,,addr
```

Where **start-addr-offset** is the offset to the starting address of the program that is being called, and **addr** is the address of the first word in the RUN UUO argument block.

Before the monitor transfers control to the program you call, it adds the program's starting address to the left half of the value in the ac and starts the program at this address. If you set the starting address offset to be anything other than Ø or 1, you are considered to be meddling with the program, unless the program being executed is an execute-only program for your job. For execute-only programs, the monitor ignores any value other than Ø or 1.

2.5.1 Functions of RUN and GETSEG

To successfully program the RUN monitor call on systems of all sizes and for programs whose size is not known at the time of the RUN monitor call's execution, you must understand the sequence of operations initiated by the RUN monitor call. Note that the RUN monitor call can be executed from either the high or the low segment.

Before calling a new program with the RUN monitor call, you should change your low segment core allocation to one page, and delete your high segment (if any).

The GETSEG and RUN monitor calls perform the following functions:

1. Using the file name you specify in the argument block, the monitor searches for a sharable high segment that is already in core that has the same name as the program you specified. If a sharable high segment already exists, it is attached to your job.

   If there is no existing sharable high segment, the monitor looks up the program on disk, searching for the same file name with the extensions .EXE, .SHR, and .HGH, in that order.

   If it finds a file with one of these extensions, the monitor loads the high segment of the file into memory, starting at the high end of your current low segment. If it does not find a file, it goes to Step 5.

   It then remaps the hiseg, placing the hiseg origin at the .JBHGH location.

   However, if the current low segment contains any pages that would conflict with the new high segment, the monitor call takes the error return, with error code 31 in the AC.

2. Then the monitor adjusts its internal data base to include the new information. Either a new user for the sharable high segment or a new high segment must be added. (If the UUO was a GETSEG, the monitor returns control to the user program at this point.)

3. Information from the vestigial job data area is copied into the low segment's job data area. The vestigial job data is always loaded into the high segment (see Chapter 4).

4. Part of the job data area is the .JBCOR word. If the left half of .JBCOR is less than or equal to 137, the monitor does not have to read data into the low segment because it is a null low segment. Then the monitor reads the right half of .JBCOR to determine how much space to allocate for the low segment. For a null low segment, control is passed to the user program at this point.

5. If the left half of .JBCOR is greater than 137, the monitor loads the program's low segment, using one of the following procedures:

   - If the original file contained both segments, the monitor continues reading the currently open file, loading the data from its low segment into memory starting at location 140. The user program is started.

   - If the original file contained only the high segment, the monitor looks up the file using the same file name and one of the extensions .EXE, .SAV, .LOW, or the extension specified in the argument block, in that order.

     If found, the file is opened and read into memory starting at location 140. The user program is started.

     If a low segment file is not found, the monitor returns control to the user, giving an error return. The monitor handles the error in either of the following ways:

     a. If the call came from the high segment, or if there is a HALT in the error return after the UUO, the monitor prints one of the following error messages:

        ```
        ?Not a save file
        ?filename.SAV not found
        ?Transmission error
        ?LOOKUP failure n
        ?nP of core needed
        ?No start adr
        ```

     b. If the call came from the low segment and there is no HALT in the error return, the monitor puts the LOOKUP error code into the ac and passes control to the user program.

The GETSEG monitor call works like the RUN monitor call, except for the following differences:

- No attempt is made to read the low segment of the file. If an .EXE file is found, only the pages representing the high segment will be merged into the user's address space.

- The only changes made to the Job Data Area are:

  a. the left half of .JBHRL is set to zero.

  b. the right half of .JBHRL is set to the highest legal high segment address.

  c. .JBSA and .JBREN in the Job Data Area are set to zero by the monitor if they point to a high segment that is being removed. If this should occur, the following message is printed on your terminal when the START or REENTER command is issued:

     ```
     ?No start adr
     ```

- If an error occurs, control is returned to the error return location, unless the left half of the error return location contains a HALT instruction; in this case, the monitor displays an error message and the program is HALTed.

- The call should be made from the low segment unless the normal return coincides with the starting address of the new high segment.
- User channels are not released. Channel Ø, however, is released because it was used by the GETSEG monitor call.
- The contents of the job's accumulators are not preserved. Therefore, any AC used as a stack pointer will become invalid.

2.5.2 Reading Command Files

Programs that accept commands can input those commands from a terminal or a file, depending on how the program is started. If a program is started at the normal starting address (.JBSA), it should display a prompt, such as an asterisk, and read commands from the terminal input buffer. With a CCL entry, commands are read from a file on disk or from a list of commands stored in memory.

CCL entry is determined by the argument to the RUN monitor call. If the RUN UUO has a Ø in the left half of the ac, the normal start address will be used. If the RUN UUO has a 1 in the left half of the ac, the CCL entry will be used; the program is started at the address found in .JBSA+1, and should read commands from TMPCOR or from an indirect command file on disk. TMPCOR is the name for the section of memory that is searched first. The format of a command file is defined by the program that must read it.

The TMPCOR area is limited in size; therefore, programs should also search for the command file on disk. If TMPCOR does not contain the command list, it should contain a file specification for an indirect command file. The file specification is usually preceded by @ to indicate indirection. (Note that the PIP program expects the @ to follow the file specification.) By convention, the file name on disk is of the form:

```
nnnabc.TMP
```

Where nnn is your job number in decimal, including leading zeros. The job number is included to allow users to run more than one job under the same PPN, and abc is usually the name looked for in TMPCOR.

For example:

```
ØØ9PIP.TMP      ;Job 9 commands to PIP
Ø39MAC.TMP      ;Job 39 commands to MACRO
```

These files are temporary and will be deleted by the LOGOUT program.

If a command file does not exist, or the program does not support CCL entry, the program should display a command prompt and accept commands from the terminal.

2.6 CONTROLLING PAGES

By using the PAGE. monitor call, you can manipulate pages in your program's virtual address space and manipulate or obtain data about those pages.

2.6.1 Handling Page Faults

When your program refers to a page of memory that is either not in physical memory or has the access-allowed bit turned off, a page fault occurs. Program control then passes to a page fault handler. The page fault handler can be either the system's internal page fault handler or your program's page fault handler, if you have defined one by setting .JBPFH in JOBDAT. (See Chapter 4 for more information on .JBPFH.) Refer to Section 2.6.3 for information on building your own page fault handler.

2.6.2 The System's Page Fault Handler

Unless your program has its own page fault handler, the monitor uses its internal page fault handler when a page fault occurs. This default page handler selects the page to swap out of memory to make room for a needed page that is not in memory.

Each page has an access-allowed bit associated with it. Periodically the page fault handler clears every page's access-allowed bit. When the page fault handler clears the access-allowed bit, a page fault occurs the next time your program references that page. After the reference is made, the monitor sets the access-allowed bit and execution of the program continues.

The system's page fault handler selects the page that has been in core the longest since the last reference to it. This selection method assures that frequently-referenced pages are likely to remain in memory, while seldom-referenced pages are likely to be paged out sooner. By using an age-ordered list and a periodic check of the access-allowed bit, the page fault handler pages on a modified first-in/first-out basis.

2.6.3 Building Your Own Page Fault Handler

You can override the system's page fault handler by setting the location .JBPFH to point to your program's page fault handler (left half = end address and right half = start address). Setting .JBPFH to zero returns your program to using the system's page fault handler. If the area of core containing your own page fault handler is destroyed, a reference to the page fault handler will result in an illegal memory reference error.

NOTE

Use the system's page fault handler if your program will execute in a non-zero section or contain thirty-bit addresses. The format given below for your own page fault handler can accept only eighteen-bit addresses, and is restricted to programs executing in Section Ø.

The format of your page fault handler must be:

Word	Symbol	Contents
Ø	.PFHJS	The instruction "JRST .+.PFHST".
1	.PFHOP	Old PC and flags. That is, the flag/PC word used for JRSTF, same as .JBTPC, .JBOPC, and so on.
2	.PFHFC	Fault word, filled in by the monitor on each page fault. The fault word is described below.
3	.PFHVT	Virtual time.
4	.PFHPR	Paging rate.
5	.PFHPV	Highest PSI vector in use (in left half); address of PSI vector (in right half).
6	.PFHUR	Version number of the page fault handler.
7-11		Reserved for runtime statistics.
12	.PFHST	Origin of the page fault handler (first instruction)

The fault word (.PFHFC) contains the following information:

Bits	Symbol	Contents
Ø	PF.HCB	Working set was changed by a routine other than this page fault handler.
1	PF.HBS	Working set has been scrambled.
2-8		Reserved for use by DIGITAL.
9-17	PF.HPN	Page number of the page causing the fault.
18-35	PF.HFC	Fault code, described below.

The fault code (PF.HFC) is one of the following:

Code	Symbol	Meaning
1	.PFHNA	Page is inaccessible (access-allowed bit is cleared) but in core.
2	.PFHNI	Page has been paged out and is not in memory.
3	.PFHUU	A page containing a monitor call argument has been paged out. This is a monitor-detected fault.
4	.PFHTI	A virtual timer trap has occurred. The monitor will cause this kind of trap every n units of time, if requested by the .STTVM function of the SETUUO monitor call.
5	.PFHZI	The page has been allocated, but is a zero page, occurring after a user instruction.
6	.PFHZU	The page has been allocated, but is zero after a monitor call is executed.

2.7 LOCKING AND UNLOCKING A JOB IN MEMORY

When a job is "locked" in memory, it is not available for swapping. You can lock a job in user memory by using the LOCK monitor call. You may want to lock a job for any of the following reasons:

- The job is a realtime job that should respond to interrupts promptly, without the delay of swapping.
- The job uses a display device that must refresh the display from a buffer without flickering.

- The job analyzes system performance, and must be invoked quickly.
- The job uses the SNOOP. UUO.

You must have the lock privilege to use the LOCK UUO. This is set by the JP.LCK bit in GETTAB table .GTPRV.

Using the LOCK call, you can lock either or both program segments into memory, and you can specify whether the memory must be physically contiguous.

A job can be unlocked (made available for swapping) using either the RESET or UNLOK. monitor call. Execution of a RESET monitor call automatically performs several other functions. You may want to unlock the job without resetting everything. You can do this by using the UNLOK. monitor call. Unlocking jobs

The UNLOK. call allows you to unlock either or both segments of your job. Note, however, that a locked high segment that is shared by several jobs will not be unlocked until the SN%LOK bit in the GETTAB table .GTSGN is off for all those jobs. The shared high segment will not be unlocked until every job sharing the segment has issued either a RESET or UNLOK. monitor call for that segment.

CHAPTER 3

JOB CONTROL

This chapter discusses initializing, starting, stopping, and suspending programs, timing considerations, and other functions pertaining to jobs.

3.1 EXECUTING A PROGRAM

After you write a program, it must be compiled and loaded before it can be executed. The compilation process depends on the programming language used in the source program, but the COMPILE monitor command will initiate compilation of any program in a supported language. Refer to the TOPS-10 Operating System Commands Manual for information about the COMPILE command and all monitor commands.

The compiled program exists on disk in relocatable format; this stage of program preparation is known as the .REL file. The LINK program (the linking loader) processes the .REL file by resolving symbol definitions and by loading the program into user core memory. After it has been loaded successfully, the program is ready to be executed or saved in its executable format. LINK offers switches to initiate execution or saving. LINK is described in detail in the TOPS-10 LINK Reference Manual.

Each time the source program is changed, it must be recompiled and reloaded before it can be executed.

Once the program is loaded into memory, execution can start immediately, but it is more common to write the core image of the program to disk, thus saving it in executable format as an .EXE file. This .EXE file can be loaded from disk and executed using the RUN monitor command.

Programs can be called from within another program as well as from monitor command level.

3.1.1 Starting a Program

You can start a program from the monitor level by using any of the following monitor commands:

- RUN loads an .EXE file from disk and starts execution at the address given by .JBSA in the job data area.
- EXECUTE starts execution of a specified program at its normal start address. This command loads the program from its .REL file. If no .REL file exists, EXECUTE compiles the source file and then loads the .REL file.

- START starts (or restarts) execution of an already-loaded program at its normal start address.
- CSTART starts execution of an already-loaded program at its normal start address, but leaves the terminal at monitor level.
- DDT starts execution of the debugging program specified by the right half of .JBDDT. (Refer to Chapter 4 for information about JOBDAT locations.)

You can continue execution of an already-loaded program from monitor level by using either of the following monitor commands:

- CONTINUE continues execution of an already-loaded program at the place where the program was interrupted.
- CCONTINUE functions like the CONTINUE command, except that it leaves your terminal at monitor level.

You can start a program from within another program with any of the following monitor calls:

- RUN transfers execution control to the specified program. The calling program is overwritten in core, and control is passed to the new program. The RUN call allows you to specify an offset to the normal starting address.
- GETSEG replaces the high segment of the calling program with a specified high segment.
- MERGE. merges specified pages of an .EXE file into the low segment of the current program.

The RUN and GET monitor commands are often used in the same session. Therefore, for these commands, the monitor saves the argument to the command (that is, the program name) in a GETTABable form. If you use one of these commands without an argument, the saved argument is assumed by the monitor. To read the argument that is currently being stored, see GETTAB tables 135-137 and 145-151.

3.1.2 Stopping a Program

You can stop execution of your program by using any of the following commands or monitor calls:

- CTRL/C, if your program is waiting for terminal input, or if your program is running but your terminal is at monitor level.
- Two CTRL/Cs, if your program is not waiting for terminal input, but is attached to your terminal.
- The HALT monitor command.
- The EXIT monitor call in your program code.
- The LOGOUT monitor call in your program code.
- The FRCUUO monitor call in your program code.

If a fatal error occurs (including a HALT), the monitor will stop your program.

3.1.3 Suspending a Program

You can suspend execution of a program until some event occurs, or until some time has elapsed, by using the following monitor calls in your program:

- The HIBER monitor call suspends execution of your program until some specified event occurs or until a specified amount of time has elapsed. The program will be continued by the monitor.
- The SLEEP monitor call suspends execution of your program until a specified time has elapsed.

You can also use the PSI system, which can interrupt a job when a particular event occurs. (Refer to Chapter 6.)

3.2 CONTROLLING MULTIPLE JOB CONTEXTS

The core image of a job, some system resources such as ENQ/DEQ locks, some terminal parameters, and monitor overhead data constitute a job's context. The CTX. UUO allows you to save and retrieve information about contexts, and manipulate them in ways that give you control over multiple jobs. For instance, using contexts, you can halt and save a running program to perform some other task, and later restore the context and continue the program.

Using CTX., you can create two kinds of contexts: inferior and parallel. Creating either an inferior or a parallel context halts the current job and saves the current context. However, when you work in an inferior context, returning to the original (superior) context causes the system to automatically delete the inferior context. When you create a parallel context, it co-exists with the original context until you explicitly delete it. You may switch between parallel contexts without deleting any of them. Under the system default, you may work with a maximum of four parallel and/or inferior contexts at any one time. This value may be changed in the user's accounting file entry.

Once the system has swapped a job's core image out to disk, the context is considered idle. The items saved in a context are:

- Program run; from physical SYS bit (JB.LSY from JBTLIM(J))
- Monitor mode bit: LDLCOM from LDBDCH(U)
- SAVCTX word in the PDB: .PDSCX(W)
- Break mask words: LDBBKM(U), LDBBKB(U), and LDBCSB(U)
- PSI data base address: JBTPIA(J)
- All IPCF-related words in the PDB
- Enqueue block chain address: .PDEQJ(W)
- Selected words in the TTY DDB
- Job status word: JBTSTS(J)

- Swapped out disk address: JBTSWP(J)
- Swapped in image size: JBTIMI(J)
- Swapped out image size: JBTIMO(J)
- High segment number: JBTSGN(J)
- Funny space (per-process space) page count: JBTPDB(J)
- Swapped out checksum: JBTCHK(J)
- Program name: JBTNAM(J)
- User PC: JBTPC(J)
- I/O wait DDB: JBTDDB(J)
- Program run data: .PDNAM(W), .PDSTR(W), .PDDIR(W), .PDSFD(W)
- Time of last reset: PDSTM(W)
- Address of user-defined commands: PDCMN(W)
- Address of UNQTAB for user-defined commands: PDUNQ(W)
- Address of DECnet session control block: PDSJB(W)

The ENQ/DEQ, IPCF, and PSI facilities can all work in conjunction with multiple contexts. The Job/Context Handle (JCH) allows a facility to uniquely identify a job and one of its contexts. JCH storage requires 18 bits. ENQ/DEQ, IPCF, and PSI have 18 bits reserved for this purpose. If you have not enabled the context facility, the JCH equals the job number. If the JCH has a zero context component, the system translates it into the JCH for the job's current context. A non-zero context component allows a program to target a job at a particular context. Refer to Chapter 22, Volume 2 for a description of the CTX. monitor call.

3.3 RUNTIMES, TIMES, AND DATES

The TOPS-10 monitor calculates runtimes, the time of day, and the date. You can obtain any of these either from your terminal or for use in your programs.

3.3.1 Runtimes

The TOPS-10 monitor has several ways of monitoring runtimes. The runtimes available to you depend on the type of processor your system uses, and on system parameters.

The KS and KL processors simulate a clock, called the APR clock, which is based on the frequency of the system power source (either 50 or 60 Hz). The APR clock may be used to keep the system time of day, because it is accurate over long periods of time. The APR clock may also be used for job accounting. However, it may not be completely accurate, as its tick may be longer than the runtime period for a job. The KS and KL processors simulate the APR clock by setting up internal timers to simulate the jiffy clock.

The DK1Ø clock has higher resolution than the APR clock, and is therefore more reliable for job accounting. High-precision runtime simulates the DK1Ø clock time.

EBOX/MBOX runtime is computed from KL1Ø accounting meters (to the nearest 1Ø microseconds). This runtime is not related to any other runtime, and it is not directly related to real time.

Runtimes returned by the monitor (by the RUNTIM call; the TIME, USESTAT, or CTRL/T monitor commands; or the .GTTIM GETTAB table) are all either high-precision runtime or EBOX/MBOX runtime (selectable with MONGEN). The type of runtime reported depends on the value of the ST%ERT bit in item %CNST2 of the .GTCNF GETTAB table. (The value 1 selects EBOX/MBOX runtime; Ø selects high-precision runtime.) The RUNTIM call reads RN.PCN in its accumulator to initiate high-precision runtime. A program that uses high-precision runtime can run successfully even though the ST%ERT bit is on. The time returned will be approximated to simulate high-precision, but the low-order bits of the high-precision word will be zero.

Monitor overhead can optionally be included in these runtimes, depending on MONGEN parameters.

3.3.2 The System Date

The DATE monitor call returns the system date as a 15-bit integer that must be decoded to obtain a calendar date. (See the DATE call in Volume 2 of this manual for an example showing how to decode the date.) This 15-bit date is in multiple-radix form, so that the difference between two dates is usually not the number of days between them. The format of the code is:

```
(day of month - 1) + 31 * [(month - 1) + 12 * (year - 1964)]
```

3.3.3 The Universal Date

The monitor also keeps the date in an alternate format, the universal date standard. This fullword value is in the form:

```
day,,fraction
```

where day is the number of days since November 17, 1858 (where November 17th is day Ø, the 18th is day 1, and so on); and fraction represents the fractional part of the day elapsed since midnight, to approximately 1/3 of a second. The fraction is the numerator of a fraction whose denominator is 2**18, so that the following expression gives the portion of the day elapsed since midnight:

```
fraction/(2**18)
```

The arithmetic difference between two universal dates gives the number of days and the portion of a day between the dates.

The universal date is stored in item %CNDTM in GETTAB table .GTCNF and is taken from the Smithsonian Universal Date/Time Standard (UDT).

To obtain the local time (at your time zone), add the contents of item %CNGMT in the same GETTAB table (.GTCNF) to UDT.

You can derive the day of the week from the UDT by dividing the left half by 7, and using the remainder to determine the day of the week, where:

0 = Wednesday, 1 = Thursday, 2 = Friday,... 6 = Tuesday

3.3.4 The System Time

You can obtain the system time in jiffies (one jiffy = 1/60 second) by using the TIMER monitor call. (For 50 Hz power supplies, 1 jiffy = 1/50 second.)

You can also obtain the system time in milliseconds (one millisecond = .001 seconds, to the nearest jiffy) by using the MSTIME monitor call. This call is preferable to the TIMER call, because the returned time is the same, regardless of the type of power supply (50 or 60 Hz).

3.3.5 Date-Time Elements from GETTAB Tables

The GETTAB table .GTCNF contains the parts of the date and time. These items are:

Symbol	Contents
%CNYER	Local year.
%CNMON	Local month.
%CNDAY	Local day of the month.
%CNHOR	Local hour.
%CNMIN	Local minute.
%CNSEC	Local second.

The cautious programmer should assume that individual items can change between successive GETTABs. If a date and time must be guaranteed to be consistent, your program should test values returned from the items above until it obtains two consecutive, identical readings, or you should derive the date and time from the UDT (available with a single GETTAB).

CHAPTER 4

THE JOB DATA AREA

Memory locations 20 through 137 (octal) and the high segment origin to hiseg origin + 7 (normally locations 400000-400007) are allocated for specific monitor and program uses. This area is called the job data area. Your program sets some locations in the job data area for the monitor's use; the monitor sets other locations for your program's use.

During a program load, LINK loads the job data area symbols if they are required to satisfy global references. In your program, refer to job data area locations by their symbol names (of the form .JBxxx). These symbols are defined as external symbols in UUOSYM.MAC, and are given values in JOBDAT.MAC.

Section 4.1 lists important JOBDAT locations. The JOBDAT locations that are not listed in this table are either unused or used only by the monitor. Your program should be written to reference only the job data area locations described in Section 4.1.

4.1 JOB DATA IN THE LOW SEGMENT

The low-segment job data area is in locations 20 through 137 (octal). Locations relevant to your job are:

Word	Symbol	Contents
40	.JBUUO	Used by hardware for processing local UUOs (opcodes 0 through 37); the hardware stores the opcode and the effective address in this location.
41	.JB41	Executed to start the user-programmed monitor call (LUUO) stored in .JBUUO; this location usually contains a JSR or PUSHJ instruction. LINK puts a HALT here if the user does not explicitly load anything else.
42	.JBERR	System program error count. The left half is reserved; the right half is the number of errors in the previous compilation.
44	.JBREL	The left half is reserved; the right half is the highest physical memory location available to the user program (low segment).
45	.JBBLT	First location of three words used by LINK to move the program before calling the exit routine. This location is used by the PA1050 program to store the return address for the user program.

Word	Symbol	Contents
74	.JBDDT	The left half is the first address after DDT, if DDT is loaded. If any other debugging program is loaded, this halfword is zero. The right half is the start address of the debugging program. .JBDDT can be set only with the SETDDT monitor call. If .JBDDT is zero, DDT has not been loaded and the monitor will attempt to read SYS:VMDDT.EXE when you execute a DDT command. If successful, VMDDT.EXE is brought into the program's virtual address space. The left and right halves of .JBDDT will be set appropriately.
74	.JBPFI	The highest location that is protected from user access. User programs cannot write into locations up to .JBPFI.
75	.JBHSO	Reserved for use by DIGITAL.
76	.JBBPT	The unsolicited breakpoint address. Use the instruction JSR @.JBBPT to invoke this facility from a program. It is necessary to explicitly search JOBDAT to define this symbol.
112	.JBEDV	The Exec Data Vector, which the monitor uses as a pointer to EDDT. This location is valid only in exec mode.
115	.JBHRL	High-segment addresses. If zero, there is no high segment. The left half is the first free location in the high segment, relative to the high-segment origin. This value is the same as the high segment length. This address is initially set by LINK and then set by the monitor on subsequent GETs, regardless of whether there is a file to initialize the low segment. The address is a relative quantity so that .JBHGH can be changed. The SAVE monitor command uses this value to determine how much to write from the high segment. The right half is the highest legal address in the high segment. This value is set by the monitor each time a user program begins execution or executes a CORE or REMAP monitor call.
116	.JBSYM	Pointer to the program symbol table created by LINK. The left half is the negative length of the symbol table, and the right half is the starting address of the symbol table. If Ø, this table does not exist. If this word is a positive number, it contains a pointer to the extended symbol table for LINK.
117	.JBUSY	Pointer to the table of undefined symbols created by LINK. The left half is the negative length of the symbol table, and the right half is the starting address of the symbol table. If .JBUSY is Ø, .JBSYM contains a pointer to an extended symbol table for LINK.

Word	Symbol	Contents
120	.JBSA	First free low-segment address and program starting address: the left half is the first free location in the program low segment, as set by LINK. The right half is the starting address of the user program, unless an entry vector is in use. (Refer to the ENTVC. monitor call in Volume 2.)
121	.JBFF	The first free address in the program's low segment. The left half is zero. The right half is the address of the first free location after the program data in the low segment. On a RESET monitor call, the value of the left half of .JBSA is moved to this location. When the monitor builds a buffer, it allocates the buffer at the address given by .JBFF, and then changes .JBFF to the address at the end of the buffer. Note that .JBFF may point to a non-legal user address if your program occupies every location in the last page of your low segment.
123	.JBPFH	Pointer to page fault handler. The left half is the last address of the page fault handler. The right half is the address of the start of the page fault handler. If this address is 0, the user program has no page fault handler; on a page fault, the monitor calls its internal page fault handler. This location remains zero when virtual under the above circumstances.
124	.JBREN	The left half is zero. The right half is the start address for the REENTER monitor command, unless an entry vector is in use. (Refer to the ENTVC. monitor call in Volume 2.) This value is set either by the user program or by LINK.
125	.JBAPR	The left half is zero. The right half is the address of a trap routine to handle APR traps. See the APRENB monitor call.
126	.JBCNI	The state of the APR as stored by the monitor instruction from a user trap.
127	.JBTPC	The PC of the next instruction to be executed after a user-enabled trap occurs. This value is set by the monitor. Use the JRSTF @.JBTPC instruction to continue execution.
130	.JBOPC	The last user-mode program counter for the job. The monitor sets this value on each DDT, REENTER, START, or CSTART monitor command. Location .JBOPC contains the effective address of the HALT instruction, when the user program contains a HALT instruction followed by the execution of a START, DDT, CSTART, or REENTER command. After an error has occurred during execution of a monitor call followed by a START, DDT, CSTART, or REENTER command, .JBOPC will point to the address of the monitor call. To resume execution, type the following command to DDT: @130$G

Word	Symbol	Contents
131	.JBOVL	The left half is the negative number (count) of the root segment overlays. The right half is the pointer to the header block for the root link of an overlay structure. You may also reference .JBOVL as .JBCHN.
133	.JBCOR	LINK-written low segment break and monitor-written SAVE or GET argument. The left half is the highest non-zero address in the program low segment, supplied by LINK. If this address is less than 140 octal, no low segment will be written by a SAVE or SSAVE monitor command. The right half is the user-specified argument for the last executed SAVE or GET command. The value in the right half is set by the monitor.
134	.JBINT	The left half is reserved. The right half is the address of the error-intercepting block. For a description of the format of the block, see Chapter 6.
135	.JBOPS	Reserved for object-time systems.
136	.JBCST	Reserved for customers.
137	.JBVER	Program version number (in octal) and flags, in the format shown below. The version number of any program can be obtained using the VERSION monitor command.

Bits	Meaning
0-2	Modifier flag:

Flag	Meaning
0	DIGITAL development group last modified the program.
1	Other DIGITAL employees last modified the program.
2-4	A customer last modified the program.
5-7	A customer's user last modified the program.

Bits	Meaning
3-11	DIGITAL's latest major revision number, usually incremented by 1 for each release.
12-17	DIGITAL's minor revision number, which is usually 0, unless the program has been modified since the last release.
18-35	The edit number, increased by 1 after each edit to the program. This value is never reset.

140	.JBDA	First location available for user program.

4.2 JOB DATA IN THE HIGH SEGMENT

Some job data area locations are in your program's high segment. These are loaded at the high-segment origin (usually 400000), so that your program actually begins at the origin + 10 (usually 400010). Refer to Section 2.4 for information about accessing the information in the high segment.

In the following description of the format of this area, all offsets are from the high-segment origin, usually .JBHGH (400000).

The format of the vestigial job data area is:

Word	Symbol	Contents
0	.JBHSA	Copy of .JBSA in the job data area.
1	.JBH41	Copy of .JB41 in the job data area.
2	.JBHCR	Copy of .JBCOR in the job data area.
3	.JBHRN	LH: used to set LH of .JBHRL RH: used to set RH of .JBREN
4	.JBHVR	Copy of .JBVER in the job data area.
5	.JBHNM	High segment name set on execution of SAVE command.
6	.JBHSM	Pointer to high segment symbol table, if any. In the left half is the length of the symbol table. In the right half is the address of the first word in the symbol table. If 0, there is no symbol table.
7	.JBHGA	GET address page number (bits 9-17).
10	.JBHDA	First word in high segment that is available to the user program.

CHAPTER 5

NETWORKS

TOPS-10 supports four types of data networks:

- ANF-10, the TOPS-10 native communications software, supports communication among DECsystem-10s and certain remote stations. ANF-10 provides terminal concentration, remote job entry, and front-end services for the monitor. Using ANF-10, you can perform data transfers between jobs running on different systems, and the same facility allows data transfers among jobs running on the same system. The ANF-10 software is bundled with the TOPS-10 monitor. This chapter discusses the monitor calls and facilities available in the ANF-10 communications environment.

- DECnet-10 allows data communication among many of DIGITAL's operating systems, for the purpose of file transfer, network virtual terminal capability, and intertask communication. (DECnet-10 must be purchased separately.) DECnet-10 is TOPS-10's implementation of the Digital Equipment Corporation Network Architecture. DECnet-10 Version 4 supports the standards defined in the DECnet Phase IV Architecture Specification, for Level 1 routing. Three monitor calls deal directly with DECnet-10:

  a. The NTMAN. call is used only by the DECnet Network Management Layer.

  b. The NSP. call is used for task-to-task program communication over DECnet network links. Section 5.3 discusses the NSP. monitor call.

  c. The DNET. call is used to access monitor-stored information about DECnet nodes, links, and networks. Section 5.4 discusses the DNET. call.

- IBM Emulation/Termination facility provides communication with IBM systems using batch job submission. IBM E/T, a separately purchased product, interacts with the GALAXY spooling and batch subsystem (Versions 4.1 and later), allowing users to submit TOPS-10 batch jobs from IBM remote stations, or to submit IBM-style batch jobs from the TOPS-10 host. This facility runs on a DN60 front end. IBM communication is described in the TOPS-10 IBM Emulation/Termination manual.

- Ethernet, which allows you to implement foreign Ethernet protocols using the ETHNT. monitor call. For example, if you wish to communicate with a printer that is on the Ethernet, but is unreachable with ANF-10 or DECnet, you could use ETHNT. to develop software that would communicate with the printer over the Ethernet. ANF-10 and DECnet-10 also function on the Ethernet hardware. Section 5.5 discusses the ETHNT. monitor call. See Volume 2 for a full description of ETHNT. functions. Refer to The Ethernet: Data Link Layer and Physical Link Layer Specifications for a description of the Ethernet hardware.

Each computer processor in a network is called a **node**. A node is either a host or a remote station. A **host node** is capable of running user programs. A **remote station**, also known as a server, is a node with more limited capabilities. It contains a software that controls specific I/O functions, such as terminal concentration, remote batch job entry, network communications links, and more.

A TOPS-10 system operates as a host node in any network. A KL-based DECsystem-10 can operate as a host node in any of the network environments described above. The RSX-20F front end is not a separate node in the network. IBM E/T is not supported on KS-based DECsystem-10s.

Each node in the network has a **node name** of six characters or less, and a **node number**, sometimes known as the **node address**. For ANF-10, the node number is octal. You can use either the node name or node number in most cases when referring to an ANF-10 node. You must refer to a DECnet node by its node name.

The terms **local node** and **remote node** distinguish the host system that is interpreting your commands (the local node) from the other nodes in the network (the remote nodes). Your terminal is automatically connected to the host system by the remote station or front end to which your terminal is physically connected.

5.1 ANF-10 NETWORK MONITOR CALLS

A MACRO program can include the following monitor calls to accomplish intertask communication in the ANF-10 network environment.

- GTNTN. UUO returns the line and remote station number of a terminal.
- GTXTN. UUO returns a terminal name for a physical node and line number.
- LOCATE UUO allows your job to send direct spooled output to the device at the node you specify.
- NODE. UUO can perform several network functions. Among them are:
  a. Assign a logical name to a device and assign the device to your job. The device may be connected to a remote system.
  b. Return a node number for a node name, or node name for node number.
  c. Send station control messages.

d. Receive boot request messages.

e. Return system configuration information (similar to the NODE command).

f. Connect or disconnect a terminal to or from a remote system.

g. List the known nodes.

h. Return node data block information.

i. Return and clear the ungreeted node flag.

- TSK. UUO allows you to use intertask communication, which may include tasks on the same system or on different systems.

- WHERE UUO returns the name of the node to which a terminal or other peripheral device is connected.

You perform I/O in the ANF-10 environment using UUOs such as IN, OUT, INBUF, OUTBUF, FILOP., and OPEN.

The following sections discuss the use of monitor calls to establish intertask communication between nodes on an ANF-10 network.

5.2 ANF-10 INTERTASK COMMUNICATION

Two tasks running in the same ANF-10 network can communicate with one another. For example, one task may wish to transfer a file to another task at another node. This communication between tasks is called **intertask communication**. Intertask communication can be used between jobs on the same system or between jobs on on different systems in the network. (In this discussion, the term **task** is used to refer to a program.)

Initially, both programs must open a channel for I/O for intertask communication. This is accomplished using the OPEN UUO with SIXBIT /TSKnn/ in the argument block (Word 1), for the device name, where nn is the node number of the node where the other task is running.

The appropriate calling sequence for opening a channel for intertask communication is:

```
        OPEN      channo,argblk
           error return
        normal return
        . . .
argblk: EXP       mode
        SIXBIT    /TSKnn/
        XWD       outblk,inblk
```

In this example, **channo** is the channel number, **argblk** is the address of the argument block. The first word of the argument block allows you to define the data mode. The I/O modes used for data transfer between tasks are:

- ASCII and ASCII line modes, in 7-bit ASCII

- Byte mode, in 8-bit bytes

- Image and image binary modes, in 36-bit words

Refer to Section 11.4 for information about I/O modes.

The second word specifies the device and node number of the other task (TSKnn).

The third word of the argument block specifies the addresses of the output (outblk) and input (inblk) buffer control blocks.

After opening the I/O channels for communicating data, the tasks must initiate the connection. The passive task describes the desired connection, and then waits for I/O to start. The active task initiates a connection and may start I/O.

Since interactive task-to-task communication is always buffered, you should use one buffer for each data request when sending data. When receiving data, use multiple buffers, so that all incoming data requests can be accommodated.

5.2.1 Initiating a Connection

The intertask connection can be initiated by either of the following methods:

- The passive task uses the LOOKUP UUO to specify the task name, and then performs an IN or INPUT. The passive task then waits for the active task to initiate I/O. The active task uses the ENTER UUO to initiate the connection, and then performs OUT or OUTPUT calls to start data transfer. This procedure is described in the following section. LOOKUP/ENTER blocks are described in more detail in Chapter 11. Either the short argument block (shown here) or the extended argument block may be used to initialize intertask communication.

- The passive task uses the TSK. UUO with the .TKFEP function code. The active task uses the TSK. function .TKFEA. Both tasks must provide Network Process Descriptor (NPD) blocks. This method of intertask communication is described in Section 5.3.1.2.

5.2.1.1 Using the LOOKUP/ENTER UUOs - After OPENing the I/O channel, the passive task must declare the task name in the LOOKUP monitor call. By specifying the task name in the argument block, the passive task ensures that only the appropriate active task can initiate I/O. The task names specified by the passive and active tasks are compared and must be equivalent before I/O can begin.

The LOOKUP calling sequence is:

```
        LOOKUP    channo,argblk
           error return
        normal return
        . . .
argblk: SIXBIT    /tsknam/
        Ø
        Ø
        ppn
```

In this sequence, **channo** is the same channel number used in the OPEN UUO, and **argblk** is the address of the argument block. In Word Ø of the argument block, **tsknam** specifies the task name in SIXBIT. Words 1 and 2 are unused. Word 3 contains the **ppn** of the active task, if different from that of the passive task. (You must have privileges to specify a PPN.) A PPN of 777777,777777 indicates full wildcard acceptance of a connection from any PPN.

The passive task can then issue an IN or INPUT monitor call for the given channel to initiate a wait state until connection is made from an active task, or, if the program has the PSI system enabled, it can act on an appropriate interrupt condition (refer to Chapter 6).

The active task uses an ENTER UUO to specify the task name for which to establish a connection. The following calling sequence is used:

```
          ENTER     channo,argblk
             error return
          normal return
          . . .
argblk:   SIXBIT    /tsknam/
          Ø
          Ø
          ppn
```

where **channo** is the channel number used in the OPEN call, and **argblk** is the address of the argument block. In the argument block, **tsknam** is the name of the task (same used in LOOKUP by the passive task. The **ppn** is the project-programmer number of the active task, which you should include only if it is different from that of the passive task. You must have privileges to specify the PPN. If you specify Ø for the PPN, it defaults to the task's own PPN.

5.2.1.2 Using the TSK. UUO - Both the passive and active tasks can use the TSK. UUO to initiate the connection. First you must OPEN an I/O channel for task-to-task communication, just as with the LOOKUP/ENTER method. However, the TSK. call gives your program greater control over the communication, and allows many functions useful in performing task-to-task communication.

TSK. UUO operates by reading a Network Process Descriptor block (NPD) for each task. In this description, the word "process" is equivalent to "task." Both tasks must specify an NPD for both the passive and active tasks. The NPD that each task specifies for the remote task must match the other task's local NPD. The format of the NPD is:

Word	Symbol	Contents
Ø	.TKNND	Node number of the task
1	.TKLNL	Length of task name
2	.TKNPN	First word of task name
	.	
	.	
n	.TKLNL+n	Last word of task name

An NPD describes a task (process). The first word (.TKNND) contains the node number of the node on which the task is running, and could be -1 to indicate any node (-1 is not valid when connecting to a remote passive task).

The second word specifies the length (in characters) of the task name. The maximum length of the task name is 100 (decimal) characters; this maximum is defined as .TKMNL in UUOSYM.MAC.

The third word contains the first word (in ASCII) of the task name. The length of the NPD, therefore, is the result of adding 2 + the contents of .TKLNL (the number of characters divided by 5 and rounded up). The wildcard characters listed here can be used in the task name:

Character	Meaning
*	Matches 0 or more characters.
?	Matches any one character.
^V	Quote next character, allowing you to specify asterisks and question marks (for wildcard characters) in the task name.

The TSK. call is invoked as shown in the following calling sequence:

```
          MOVE    ac,[XWD length,argblk]
          TSK.    ac,
             error return
          normal return
argblk:   EXP     function-code
          EXP     channo
          EXP     locnpd
          EXP     remnpd
```

In the TSK. calling sequence, the ac is first loaded with a word consisting of the length of the argument block (in this case, 4) and the address of the first word in the argument block (argblk). The argument block contains the following words.

In Word 0, the function code is indicated by **function-code.** For the passive task, (similar to LOOKUP, above) this is .TKFEP. For the active task, (similar to ENTER, above) the function code is .TKFEA. Although a monitor call must be issued by each task, both tasks need not use the same calling procedure. That is, one task may initiate the connection using a LOOKUP or ENTER call, and the other task may use the appropriate TSK. function.

In Word 1, the channel number is indicated by **channo.** This is identical to the channel number used in the OPEN UUO.

In Word 2, the address of the local Network Process Descriptor, indicated here by **locnpd.** This is the task's own NPD.

In Word 3, the address of the remote Network Process Descriptor, indicated here by **remnpd.** For the passive task, this is the NPD of the task from which it will accept a connection. For the active task, this is the NPD of the task with which to attempt a connection.

After using the TSK. UUO with the .TKFEP function, the passive task can wait for input from the active task.

After using the TSK. UUO with the .TKFEA function, the active task can begin data transfer.

5.2.2 Sending and Receiving Between Tasks

When task-to-task communication is established with LOOKUP/ENTER calls, the monitor generates NPDs for the tasks. Thus, the programs can use TSK. call functions even though communication was not initiated with the TSK. monitor call. The monitor generates a local NPD and a remote NPD for each program.

The local NPD of each task consists of the node number of the local node (the node on which it is running), and the task name consists of the program name (obtained from .GTNAM) and the task's [PPN].

The remote NPD for each task consists of the node number generated from the OPEN call, where **TSKnn** would contain the node number as **nn**, or -1 if only **TSK** was specified (implying that connections from any node will be accepted). The task name would be generated by the file specification in the LOOKUP/ENTER block.

Because of this facility, two tasks can communicate using different calling procedures. One task can use the appropriate LOOKUP or ENTER call, and the other task can use the complementary TSK. function. Note, however, that the TSK. call must include references to NPDs that are equivalent to those generated by TSKSER for LOOKUP/ENTER sequences. The LOOKUP/ENTER NPD contains a file specification for the task name. This must be matched exactly by the program issuing the TSK. call.

When the intertask communication is established, the two tasks can send and receive data using the normal I/O monitor calls (IN, INPUT, OUT, and OUTPUT) for the open channel.

Your program should require the communicating tasks to verify the intertask communication by sending and receiving identifying codes; these codes should be unique among tasks on the network so that no mistaken communication occurs.

The TSK. call offers several functions useful for performing I/O as well as establishing the task-to-task link. The TSK. functions are:

Fcn-code	Symbol	Meaning
1	.TKFRS	Returns the state of the communications link specified in the argument block. The link can be idle, waiting for a connection, waiting for a connect confirmation, active, or waiting for a disconnect confirmation.
2	.TKFEP	Creates a passive link.
3	.TKFEA	Creates an active link.
4	.TKFEI	Enters idle state. This allows the monitor to CLOSE the connection.
5	.TKFWT	Enters wait state.
6	.TKFOT	Outputs messages with control of message disassembly.
7	.TKFIN	Inputs messages with control of message reassembly.
1Ø	.TKFRX	Returns the state of the link and the maximum data message size.

5.2.3 Breaking the Intertask Communication

Either the active task or the passive task can break the intertask communication. When one task issues a CLOSE monitor call for the communication channel, the other task receives an end-of-file on its channel. When the task issues a RELEAS monitor call for the channel, the communication is completely broken. Both tasks should close and release their channels.

5.3 TASK TO TASK PROGRAMMING WITH DECnet-1Ø

You can write MACRO programs to communicate with tasks on another DECnet node. When doing so, you use the NSP. monitor call to interface to DECnet-1Ø. This section describes the functions of the NSP. monitor call. These functions allow you to:

- Declare a network task as willing to accept connections.
- Initiate a request for a connection to another network task.
- Accept or reject a request for a connection from another network task.
- Transmit data to and receive data from another network task.
- Request the status of a logical link.
- Read the connect attributes of a network task.
- Exchange interrupt messages with other network tasks.
- Set buffer quotas for the link.
- Set a reason mask for PSI interrupts.
- Disconnect a network connection.

The basic form of the NSP. monitor call is:

```
     MOVEI   AC,ADDR              ;Arg block pointer
     NSP.    AC,                  ;Do the UUO
       Error return               ;AC contains error code
     Normal return                ;UUO succeeded, AC unchanged
```

The AC always points to a data block that has the general form:

Word	Symbol	Contents	
		Ø – 17	18 – 35
Ø	.NSAFN	Flags+Fcn	Length
1	.NSACH	Status	Channel #
2	.NSAA1	First Argument	
3	.NSAA2	Second Argument	
4	.NSAA3	Third Argument	

On an error (non-skip) return, AC always contains an error code. The error codes and their meanings are listed in Volume 2, in the description of the NSP. UUO. On a normal (skip) return, the AC is unchanged.

You may set two flags in the NSP. monitor call.

- NS.WAI (bit Ø of .NSAFN) indicates whether the program should wait for the function to be completed before returning from the monitor call. If the bit is set, the monitor waits.

- NS.EOM (bit 1 of .NSAFN) indicates whether an end of message is to be sent with a message. DECnet-1Ø returns the NS.EOM flag as appropriate on a data read. If the program sets NS.EOM on a normal data read call, DECnet-1Ø truncates data that overflows the program's buffer.

The NSP. UUO has the functions listed below. The function code must be in Bits 9 through 17 of the first word of the argument block (.NSAFN). The NSP. status variable and channel number are always the second word (.NSACH). All functions return the after-function status of the link in the left half of .NSACH. All functions ignore the status in .NSACH when reading arguments, so the program can pass an old status along with a channel number if that is convenient. The interpretation of the argument words varies with the function.

Table 5-1: NSP. UUO Functions

Fcn-code	Symbol	Meaning
1	.NSFEA	Enter Active State
2	.NSFEP	Enter Passive State
3	.NSFRI	Read Connect Information
4	.NSFAC	Accept Connect
5	.NSFRJ	Reject the Connect
6	.NSFRC	Read Confirm Information
7	.NSFSD	Synchronous Disconnect
1Ø	.NSFAB	Abort and Release
11	.NSFRD	Read Disconnect Data
12	.NSFRL	Release Channel
13	.NSFRS	Read Channel Status
14	.NSFIS	Send Interrupt Data
15	.NSFIR	Receive Interrupt Data
16	.NSFDS	Send Normal Data
17	.NSFDR	Receive Normal Data
2Ø	.NSFSQ	Set Quotas
21	.NSFRQ	Read Quotas
22	.NSFJS	Set Job Quotas
23	.NSFJR	Read Job Quotas
24	.NSFPI	Set PSI Conditions

5.3.1 Specifying a Destination Task

A task declares that it is ready to accept a connection by executing the Enter Passive function (.NSFEP). The .NSFEP argument list has the format:

```
.NSAFN    Flags+XWD .NSFEP,3
.NSACH    XWD Status, Channel number (assigned by this call)
.NSAA1    Connect block pointer
```

The link is put into Connect Wait state (.NSSCW) and remains in this state until a connect initiate message is received that matches the connect block, or until the link is released.

DECnet-1Ø uses the connect block specified by the connect block pointer (.NSAA1) as a pattern for incoming connect initiate messages. The connect block has fields for pointers to source and destination task descriptor blocks, and pointers to string blocks for the node name, user-id, password, account, and user data. Fields that are zero in the connect block are considered wildcards and match anything. The only field that must be specified is the pointer to the destination task descriptor.

The connect block has the following format:

Word	Symbol	Field	Type
Ø	.NSCNL	Ø | Length	length of this argument block
1	.NSCND	Node name	pointer to string block (max 6 bytes)
2	.NSCSD	Source	pointer to task descriptor block
3	.NSCDD	Destination	pointer to task descriptor block
4	.NSCUS	User-id	pointer to string block (max 39 bytes)
5	.NSCPW	Password	pointer to string block (max 39 bytes)
6	.NSCAC	Account	pointer to string block (max 39 bytes)
7	.NSCUD	User Data	pointer to string block (max 16 bytes)

The connect block contains pointers to other blocks, such as those containing the node name or accounting information, and those containing information describing the source and destination tasks.

The blocks for the node name, user-id, password, account, and data are called string blocks. A string block has the following format:

Word	Symbol	Field	Type
Ø	.NSASL	NS.ASC,NS.ASL	LH: byte count; RH: block length in words
1	.NSAST	c1,c2,c3,c4,Ø	8-bit bytes of data
	NS.ASL-1	cn-1,cn,Ø	

The user-id, password, account, and user data are optional. They can be used by the destination task to validate a network connection or to perform any other handshaking functions agreed to by both tasks. Except for the data (which can be up to 16 characters long), these strings can be up to 39 characters long. They must consist of alphanumeric ASCII characters (including the hyphen, dollar sign, and underscore).

The task descriptor block identifies the source or destination task. Its format is shown below:

Word	Symbol	Field	Type
Ø	.NSDFL	Ø , Length	length of the argument block
1	.NSDFM	Format type	Ø, 1 or 2
2	.NSDOB	Object type	integer
3	.NSDPP	PPN	2 half words (16 bits, 16 bits)
4	.NSDPN	task name	pointer to string block (max 16 bytes)

By including the proper combination of values in the task descriptor block, you identify the task and specify whether it is a system program or a user program. Table 5-2 shows the information you must supply for each type of program.

Table 5-2: Allowable Combinations of Task Descriptor Values

Kind of Program	Format Type	Object Type	PPN	Task Name
DECnet system program	Ø	1-127	Ø	Ø
Customer system program	Ø	128-255	Ø	Ø
User program (name only)	1	Ø	Ø	string pointer
User program (with PPN)	2	Ø	n,n	string pointer

Use format type Ø only with a non-zero object type to specify a system program. Object types 1 through 127 identify DECnet utilities or control programs. Only a privileged program can have an object type of 1 through 127. Object types 128 through 255 identify customer system programs and are assigned by the system manager. A program need not be privileged to have an object type of 128 through 255. Refer to UUOSYM.MAC for a list of the currently-defined object types.

Use Format types 1 and 2 to specify user programs. With Format types 1 and 2, you must specify an object type of Ø and a pointer to the task name. With Format type 1, you specify a task name of up to 16 characters, but not a project-programmer number. Format type 2 allows you to specify both a task name and a project-programmer number, but there are restrictions on each. The task name can only be up to 12 characters long because DECnet-1Ø adds the project-programmer number to it. Also, your project-programmer number must fit into 16 bits, or your program must be privileged (so it does not require a PPN). Otherwise, your program must identify itself using format type 1.

When DECnet-1Ø receives a connect initiate message, it matches the message against the connect blocks of successive destination tasks until it finds a match. When a match succeeds, DECnet-1Ø puts the link in the Connect Received (.NSSCR) state, and passes control to the destination program. At this point the program can read the connect data and accept or reject the connection.

Example of Specifying a Destination Task

This example shows use of the NSP. UUO to declare that a program is a destination task. It also shows the connect, task descriptor, and string blocks for the Enter Passive function.

```
; Begin by using the NSP. Enter Passive function to wait
; for a program on some other host to send a message.
ENTPAS:
        MOVEI   T1,EPBLK        ; Pointer to Enter Passive arg block
        NSP.    T1,             ; Enter Passive
          JRST  [OUTSTR [ASCIZ /?Can't Enter Passive /]; Error
                 JRST    EREXIT] ; return

; Argument block for Enter Passive
EPBLK:  NS.WAI+XWD  .NSFEP,3    ; Wait bit, function, block length
        XWD     Ø,Ø             ; No status or channel number yet
        XWD     CONBLK          ; Pointer to Connect Block

; Connect block for Enter Passive
CONBLK: EXP     4               ; Length of block in words
        EXP     Ø               ; Accept connects from any host
        EXP     Ø               ; Accept connections from any source
        EXP     TDB1B           ; Destination task descriptor block

; Destination task descriptor block
TDB1B:  EXP     5               ; Length of TDB
        EXP     2               ; Format type 2
        EXP     Ø               ; Object type Ø
        XWD     2Ø,72ØØ         ; Project-programmer number
        EXP     STRB1C          ; String block for destination name

; String block for destination name
STRB1C: XWD     4,2             ; Number of bytes, number of words
        BYTE    (8) "P", "G", "M", "B"
```

5.3.2 Specifying a Source Task

A task declares that it is a source program by executing the Enter Active function (.NSFEA). The .NSFEA argument list has the format:

```
.NSAFN    Flags+XWD .NSFEA,length (length = 3 to 5)
.NSACH    XWD Status, Channel number (assigned by this call)
.NSAA1    Connect block pointer
.NSAA2    Segment size (optional; default: maximum allowed)
.NSAA3    Flow control (optional, privileged; default: segment)
```

DECnet-1Ø sends a connect initiate message using the information in the connect block pointed to by .NSAA1, and the link enters the Connect Sent state (.NSSCS). The source connect block has the same format as that used for a destination task.

When DECnet-1Ø is transmitting data across a physical link, the data is in the form of segments whose maximum size is set during network generation. The actual segment size used for a particular logical link is negotiated by the two hosts when the link is being set up. So, when transmitting a message, DECnet-1Ø will try to fit it into a segment, breaking it if the message is larger than a segment, or sending a partial segment if the message is smaller than a segment.

To enhance performance, you may wish to find out the segment size and set the program's buffers to that size. Then each segment transmitted would contain a complete message. The program can find the segment size of a logical link by executing the Read Channel Status function.

Flow control is necessary because it may take some time for a message segment to travel through the network from the source node to the destination node. Therefore, it is desirable for a node to be able to transmit a number of segments, one after another, without waiting for an acknowledgement of one segment before transmitting another. DECnet recognizes three types of flow control:

- Segment flow control (.NSFCS) - The receiving node sends a request for data that includes the maximum number of message segments it can accept at one time. This is the default for DECnet-1Ø.

- Message flow control (.NSFCM) - The sending node transmits all the segments necessary to form a complete message. DECnet-1Ø supports this type of flow control for sending messages.

- No flow control (.NSFCØ) - A node sends segments without waiting for a data request from the receiving node. When the receiving node fills its buffers and cannot handle any more segments, it sends an OFF link service message to the sending node to stop the flow. The sending node stops sending and will not send any more segments until the receiving node signifies that it can again accept segments by sending an ON.

If the connection attempt succeeds, DECnet-10 sets the link state to Running (.NSSRN); if the attempt fails, DECnet-10 sets the link state to Connect Rejected (.NSSRJ).

Example of Specifying a Source Task

```
; Begin by using the Enter Active function to attempt to establish
; a logical link with PGMB on system HOSTB.
        MOVEI   T1,EABLK        ; Point to Enter Active arg block
        NSP     T1,             ; Enter Active
          JRST  [OUTSTR [ASCIZ /?Can't Enter Active /]; Error
                 JRST    EREXIT] ; return

; Argument block for Enter Active

EABLK:  NS.WAI+XWD .NSFEA,3     ; Wait bit, function, block length
        XWD     0,0             ; No status or channel number yet
        XWD     CONBLK          ; Pointer to connect block

; Connect block for Enter Active
CONBLK: EXP     4               ; Length of block in words
        EXP     STRB1A          ; Pointer to string block of node name
        EXP     TDB1A           ; Source task descriptor block
        EXP     TDB1B           ; Destination task descriptor block

; String block for node name
STRB1A: XWD     5,3             ; Number of bytes, number of words
        BYTE    (8) "H", "O", "S", "T", "B" ; Name of destination node

; Source task descriptor block
TDB1A:  EXP     5               ; Length of TDB
        EXP     2               ; Format type 2
        EXP     0               ; Object type 0
        XWD     20,7100         ; Project-programmer number on HOSTA
        EXP     STRB1B          ; String block for source name

; String block for source name
STRB1B: XWD     4,2             ; Number of bytes, number of words
        BYTE    (8) "P", "G", "M", "A"

; Destination task descriptor block
TDB1B:  EXP     5               ; Length of TDB
        EXP     2               ; Format type 2
        EXP     0               ; Object type 0
        XWD     20,7200         ; Project-programmer number on HOSTB
        EXP     STRB1C          ; String block for destination name

; String block for destination name

STRB1C: XWD     4,2             ; Number of bytes, number of words
        BYTE    (8) "P", "G", "M", "B"
```

5.3.3 Reading the Connect Information

After DECnet-10 has matched the source and destination tasks, it puts the link into Connect Received state. The destination task can accept or reject the connection at this point, or it can execute the Read Connect Information function (.NSFRI) to move the connect initiate data into the connect block supplied in the call. The program can then examine the data to decide whether to accept or reject the connection. The .NSFRI argument list has the format:

```
.NSAFN    Flags+XWD .NSFRI,length (length = 3 to 5)
.NSACH    XWD Status, Channel number
.NSAA1    Connect block pointer
.NSAA2    Segment size (optional)
.NSAA3    Flow control (optional)
```

The program must specify a pointer to each block. However, it can specify the length of the block as zero, and DECnet-10 ignores the data. If the program uses 0 instead of a pointer, DECnet-10 accepts it as the pointer to AC 0 and stores the data starting at AC 0.

If the destination program has included fields for segment size and flow control, DECnet-10 stores those values for the source node.

The UUO takes the error return if the link is not in one of the following states:

```
.NSSCR    Connect Received
.NSSCW    Connect Wait
.NSSRN    Running
```

If the link is in Running state and the program has issued neither read nor write functions, DECnet-10 passes the connect initiate data to the program.

Example of Reading the Connect Information

```
; When control reaches this point, a program is attempting to initiate
; a connection with this program. The Read Connect Info function
; determines information about the job trying to establish contact.
EVALCN:
        HRRM    CHAN,RIBLK+.NSACH; Store channel number into arg block
        MOVEI   T1,RIBLK        ; Point to Read Connect Info arg block
        NSP.    T1,             ; Read Connect Info
          JRST  [OUTSTR [ASCIZ /?Can't Read Connect Info /]; Error
                 JRST    EREXIT] ; return

; Here the program checks to make sure that the source PPN is [20,*].

        HLRZ    T1,SRCPDB+.NSDPP; Get left half of source PPN field
        CAIE    T1,20           ; Test for project number 20
        JRST    REJCON          ; Reject connection if not
        JRST    ACCCON          ; Accept connection if so

; Argument block for Read Connect Info
RIBLK:  NS.WAI+XWD  .NSFRI,3    ; Wait bit, function, block length
        XWD     0,0             ; Code supplies channel number
        XWD     SRCCNB          ; Pointer to source connect block
```

```
; Connect Block for Read Connect Info
SRCCNB: EXP     ^D8             ; Length of block in words
        EXP     STNODE          ; String block for node name
        EXP     SRCTDB          ; Source task descriptor block
        EXP     DSTTDB          ; Destination task descriptor block
        EXP     STUSID          ; String block for user id
        EXP     STPSWD          ; String block for password
        EXP     STACCT          ; String block for account
        EXP     STDATA          ; String block for user data

; String block for node name
STNODE: XWD     0,3             ; Number of words -- max 6 bytes
        BLOCK   2

; Source task descriptor block
SRCTDB: XWD     0,5             ; Number of words
        EXP     0,0,0           ; Format type, Object type, PPN
        EXP     STNAME          ; String block for task name

;Destination task descriptor block
DSTTDB: XWD     0,0             ;Block is zero-length; no info wanted

; String block for user id
STUSID: XWD     0,^D11          ; Number of words -- max 39 bytes
        BLOCK   ^D10

; String block for password
STPSWD: XWD     0,^D11          ; Number of words -- max 39 bytes
        BLOCK   ^D10

; String block for account
STACCT: XWD     0,^D11          ; Number of words -- max 39 bytes
        BLOCK   ^D10

; String block for data
STDATA: XWD     0,5             ; Number of words -- max 16 bytes
        BLOCK   4

; String block for source name
STNAME: XWD     0,5             ; Number of words -- max 16 bytes
        BLOCK   4
```

5.3.4 Accepting the Connection

Once the link is in Connect Received state, the destination task can accept or reject the connection, whether or not it reads the connect initiate information. By executing the Accept Connect function (.NSFAC), the destination task declares that it will exchange data with the source task. The .NSFAC argument list has the format:

```
.NSAFN    XWD .NSFAC,length (length = 2 to 5)
.NSACH    XWD Status, Channel number
.NSAA1    Pointer to user data (string block pointer, optional)
.NSAA2    Segment size (optional; default: maximum allowed)
.NSAA3    Flow-control (optional, privileged; default: segment)
```

If execution of the Accept Connect function succeeds, DECnet-10 sends a connect confirm message to the source task and puts the link in Running state (.NSSRN). The connect confirm message contains the optional data supplied by the destination task, the segment size agreed to by both nodes, and the flow control to be used by the destination node.

If the link is not in Connect Received state when the Accept Connect function executes, the UUO takes the error return.

Example of the Accept Connection Function

```
; The program comes here to accept the requested connection
ACCCON:
        HRRM    CHAN,ACBLK+.NSACH; Store channel number into arg block
        MOVEI   T1,ACBLK        ; Point to Accept Connection arg block
        NSP.    T1,             ; Accept Connection
          JRST  [OUTSTR [ASCIZ /?Can't Accept Connection /]; Error
                JRST    EREXIT] ; return

; Argument block for Accept Connection
ACBLK:  NS.WAI+XWD  .NSFAC,2    ; Wait bit, function, block length
        XWD     0,0             ; Code supplies channel number
```

5.3.5 Rejecting the Connection

Once the link is in Connect Received state, the destination task can accept or reject the connection, whether or not it reads the connect initiate information. By executing the Reject Connect function (.NSFRJ), the destination task declares that it will not exchange data with the source task. The .NSFRJ argument list has the format:

```
.NSAFN    XWD .NSFRJ,length (length = 2 or 3)
.NSACH    XWD Status, Channel number
.NSAA1    String block pointer to user data (optional)
```

If the link is not in Connect Received state upon execution of the Reject Connect function, the UUO takes the error return.

Example of the Reject Connection Function

```
; The program comes here to reject the requested connection
REJCON:
        HRRM    CHAN,RJBLK+.NSACH; Store channel number into arg block
        MOVEI   T1,RJBLK        ; Point to Reject Connection arg block
        NSP.    T1,             ; Reject Connection
          JRST  [OUTSTR [ASCIZ /?Can't Reject Connection /]; Error
                JRST    EREXIT] ; return

; After it has rejected the connection, the program goes back and
; executes the Enter Passive function again
        JRST    ENTPAS

; Argument block for Reject Connection
RJBLK:  NS.WAI+XWD  .NSFRJ,2    ; Wait bit, function, block length
        XWD     0,0             ; Code supplies channel number
```

5.3.6 Reading the Connect Confirm Data

If the destination task accepts the connection, the DECnet software at the destination node sends a connect confirm message to the DECnet software at the source node. The source task can (optionally) read the data in the connect confirm message by executing the Read Connect Confirm Data function (.NSFRC). The .NSFRC argument list has the format:

```
.NSAFN    Flags+XWD .NSFRC,length (length = 2 to 5)
.NSACH    XWD Status, Channel number
.NSAA1    String block pointer to user data (optional)
.NSAA2    Segment size (optional)
.NSAA3    Transmit flow control mode (optional)
```

If the link is in Running (.NSSRN) state but the program has not executed any read or write functions on this link, the UUO returns the connect confirm data. However, if the program has executed a read or write function on this link, DECnet-10 discards the connect confirm data, and the UUO takes the error return. If the link is in any state other than Connect Sent (.NSSCS) or Running (.NSSRN), the UUO also takes the error return.

5.3.7 Reading the Status of the Link

The program can check the status of an assigned channel (and therefore the link) by executing the Read Channel Status function (.NSFRS). The .NSFRS argument list has the format:

```
.NSAFN    XWD .NSFRS,length (length = 2 to 4)
.NSACH    XWD Status, Channel number
.NSAA1    Segment size for this link (optional)
.NSAA2    XWD Remote flow control, local flow control (optional)
```

The left half of the second argument (.NSACH) contains the status variable, which contains the following fields:

Table 5-3: Fields in .NSACH (status variables)

Bits	Symbol	Meaning
0	NS.IDA	Single bit. If set, there is interrupt data available to be read.
1	NS.IDR	Single bit. If set, the task can send interrupt data.
2	NS.NDA	Single bit. If set, there is normal data available to be read.
3	NS.NDR	Single bit. If set, the task can send normal data.
12-17	NS.STA	6-bit field that contains the state of the connection. The state values are listed in Table 5-4.

The four data flags are set only if the link is in an appropriate state for the indicated functions. Table 5-4 lists the NSP. UUO connection states.

Table 5-4: NSP. Connection States

Code	Symbol	Meaning
1	.NSSCW	Connect wait: The task has executed the Enter Passive function and is awaiting the receipt of a connect initiate message.
2	.NSSCR	Connect Received: The task has executed the Enter Passive function and has received a connect initiate message; it may now read the connect data and must either accept or reject the message.
3	.NSSCS	Connect Sent: The task has performed an Enter Active function which sent a connect initiate message, and is now awaiting either a connect confirm (and entry into the Running state) or a connect reject (and entry into the Reject state).
4	.NSSRJ	Reject: The remote node has rejected this node's connect initiate attempt. The task should read the disconnect data and release the channel.
5	.NSSRN	Running: The link is up and may be used for the transfer of data.
6	.NSSDR	Disconnect Received: The task has received a disconnect initiate message. The task should read the disconnect data and release the channel.
7	.NSSDS	Disconnect Sent: The task has performed a Synchronous Disconnect function and is awaiting a disconnect confirm. During this time, the task should be prepared to read data from the link (the other end having not yet received the disconnect), but may not use the link for the transmission of new data.
10	.NSSDC	Disconnect Confirmed: This state is entered from the Disconnect Sent state when the disconnect is finally confirmed. At this point, the only legal functions are Release and Read Status. The task should release the channel.

Table 5-4: NSP. Connection States (Cont.)

Code	Symbol	Meaning
11	.NSSCF	No Confidence: DECnet has no confidence in this logical link because the remote node is not acknowledging messages. The local node has retransmitted a message more than the 'retransmit factor' number of times without receiving an acknowledgment. The retransmit factor is a system parameter set by the system manager. The task should release a channel in this state.
12	.NSSLK	No Link: There is no link because the remote node no longer recognizes this logical link. This can happen if the remote node is reloaded quickly, or if the remote task is aborted without sending a disconnect initiate message for some reason. The task should release a channel in this state.
13	.NSSCM	No Communication: There is no communication between this node and the remote node. A connect initiate cannot succeed because there is no communication with the requested node. This state can only be entered from Connect Sent state. The task should release a channel in this state.
14	.NSSNR	No Resources: This state is entered from Connect Sent state when a No Resources message is received from the destination node, which had insufficient resources to make the requested connection. The task should release a channel in this state.

Other functions can provide the same information as .NSFRS. However, a program could use the Read Channel Status function if it were a destination node that required the segment size. It could also use this function if it became uncertain of the link status (perhaps because it had not recently received data).

Example of Read Link Status

```
; If there's a channel number, read the status of the channel and
; analyze it
REDSTS:
        HRRM    T2,RSBLK+.NSACH  ; Store channel number into arg block
        MOVEI   T1,RSBLK         ; Point to Read Status arg block
        NSP.    T1,              ; Read Status
          JRST  [OUTSTR [ASCIZ /?Can't Read Status /]; Error
                 JRST    TYPRET] ; return

; Argument block for Read Status
RSBLK:  XWD     <(NS.WAI)>+.NSFRS,2 ; Wait bit, function, block length
        XWD     0,0              ; Code supplies channel number
```

5.3.8 Using the PSI System

A task source or destination task can be programmed to perform intertask I/O synchronously or asynchronously. With synchronous programming, the task sets a bit, called the wait flag (NS.WAI), so that each time the task executes the NSP. UUO it waits (blocks) until the I/O has been entirely completed. With asynchronous programming, the task does not set the wait bit so that the task can continue executing even if the NSP. function has not completed. The task must check the status variable to determine when the requested function is completed, or use the PSI (Programmed Software Interrupt) system so that it will be interrupted when the status changes, indicating that the NSP. function is completed. However, the program must enable the PSI system and set a PSI reason mask before DECnet-1Ø can cause an interrupt.

5.3.9 Setting the PSI Reason Mask

The PSI reason mask is an 18-bit value that contains fields corresponding to the fields in the DECnet-1Ø status variable. A program can set this mask by executing the Set PSI Reason Mask function (.NSFPI). The .NSFPI argument list has the format:

```
.NSAFN    Flags+XWD .NSFPI,3
.NSACH    XWD Status, Channel number
.NSAA1    XWD Ø,reason mask
```

In the right half of the third argument, the program sets to 1 those bits that correspond to the status or states that will cause an interrupt. All DECnet programmed interrupts come through a single PSI interrupt condition. You cannot assign a different interrupt to each DECnet channel, as you can for normal TOPS-1Ø I/O.

When a program executes the Set PSI Reason Mask function, DECnet-1Ø simulates a status change from zero to the current status for the affected link. Thus, if the program has enabled the PSI system and set that DECnet-1Ø condition in the reason mask, when the program executes the .NSFPI function, DECnet-1Ø causes a PSI interrupt. This "free" interrupt enables a PSI-driven program to do all its DECnet checking in the PSI routine.

For any bit set in the reason mask corresponding to a status (a single-bit field), only changes from false to true cause PSI interrupts. For example, Normal Data Available changing from false to true causes an interrupt, but not vice versa. For the bit fields set in the reason mask corresponding to a state (more than one bit in the field), all changes cause PSI interrupts. For example, changing the state from .NSCCS (Connect Sent) to .NSCRN (Running) means that a connect confirm message has arrived.

5.3.10 Enabling the PSI Interface

The program can enable the PSI system for any DECnet-10 channel by doing the following:

```
        MOVEI   AC,IVB          ;address of Interrupt Vector Block
        PIINI.  AC,             ;initiate PSI system
                error return
        normal return

        MOVE    AC,[PS.FON!PS.FAC,PSIARG]
        PISYS.  AC,             ;enable PSI
                error return
        normal return
        ...

        MOVEI   AC,NSPARG
        NSP.    AC,             ;NSP. argument list
                error return    ;to specify reason for interrupt
        normal return

IVB:    ...                     ;interrupt vector: allow one
                                ;4-word control block per channel
ICB:    EXP HANDLER             ;interrupt control block
        0                       ;specifies address of code
        0                       ;to handle interrupt
        0

        ...

PSIARG: EXP .PCNSP              ;NSP.-type interrupt
        XWD <ICB-IVB>,0         ;offset in IVB to ICB
        XWD priority,0          ;priority of NSP. interrupts

NSPARG: XWD .NSFPI,3            ;function code length
        XWD 0,channel           ;status word (status,channel number)
        XWD 0,reason mask       ;reason word

HANDLER: ...                    ;code to handle NSP. interrupt
```

When DECnet-10 interrupts the program, the status word of the interrupt control block contains:

```
     XWD status, channel
```

Note that the PISYS. status word has the same format as the second argument (.NSACH word) of the NSP. function argument block.

A program that has several links open at one time can include tables indexed by the channel numbers of the links. DECnet-10 assigns consecutive channel numbers starting with the lowest number available. Note, however, that DECnet-10 also reassigns channel numbers if the program releases channels; therefore, the channel numbers may not be sequential according to the order that the program first opened the channels.

If the status word for a DECnet-10 PSI interrupt is zero, the program should ignore the interrupt.

5.3.11 Reading and Setting the Link Quota and Goal

The DECnet-10 administrator allocates monitor buffers that DECnet-10 uses to hold message segments being sent or received, and also sets a default number of buffers to use for each logical link. The Set Quota function allocates a portion of those buffers to a particular link. The Set Quota function also allows the program to set the percentage of buffers allocated to a link for input (receiving).

If the program has privileges, it can also use the Set Quota function to establish the data request input goal. DECnet-10 uses segment flow control, in which the receiving node must request data before the sending node can send it. To keep message segments flowing smoothly, DECnet-10 can be asked to send data requests before the receiving program issues a read request. DECnet-10 queues message segments that arrive before the receiving program issues a read function for them.

The input goal controls the number of data requests that DECnet-10 will try to keep outstanding at the remote node. Whenever DECnet-10 receives a message segment, it will send enough data requests to the remote node to bring the total outstanding data requests to the goal.

If the input goal is larger than the link quota, this creates a pool of "cached" messages that have been received but not yet acknowledged ("committed"). DECnet allows cached messages to be discarded at any time because the source node will resend them after a timeout.

To change the link quota, percent input, or input goal (if privileged), the program can execute the Set Quota function (.NSFSQ). The .NSFSQ argument list has the format:

```
.NSAFN    Flags+XWD .NSFSQ,length (length = 3 to 5)
.NSACH    XWD status, channel number
.NSAA1    Link quota
.NSAA2    Percent of input (optional)
.NSAA3    Input goal (privileged optional)
```

The program can set the link quota for each link to any value up to the maximum number of buffers in the pool. DECnet-10 will allocate that number of buffers but will not necessarily use them all for that link.

The percent input can optionally be set to any number between 0 and 100; the default is 50.

To find out the number of buffers allocated to the link, the percentage of those buffers allocated for input, and the input goal, the program can execute the Read Quota function (.NSFRQ). The .NSFRQ argument list has the format:

```
.NSAFN    Flags+XWD .NSFRQ,length (length = 3 to 5)
.NSACH    XWD status, channel number
.NSAA1    Link quota
.NSAA2    Percent of input (optional)
.NSAA3    Input goal (privileged, optional)
```

If the argument block contains five words, all the values (quota, percent input, and input goal) will be returned.

5.3.12 Transferring Information Over the Network

Once a network connection has been established, the task at either end of the logical link can send information to the task at the other end. DECnet-1Ø provides functions for data messages and interrupt messages. Data messages are primarily used by network tasks to move blocks of data. Interrupt messages are used by network tasks to exchange small amounts of data (16 bytes or less) that are not sequentially related to the main data flow.

Data transfers over a logical link involve the segmenting and reassembling of data at both the logical and physical link levels. The network software accepts data from the user program, segments it to conform to the maximum segment size allowable on that logical link, precedes each segment with a header, and passes these segments to the physical link management layer. This layer segments the data to conform to the maximum segment size allowable on the physical link and proceeds each segment with a header to form a packet. These packets are then sent over the physical line to the destination node. At the destination node the reverse procedure takes place: headers are stripped and segments reassembled.

Occasionally, errors or status changes in a task require bypassing the normal flow of data so the message is delivered promptly. DECnet-1Ø allows for the transmission and reception of short messages called interrupt messages. An interrupt message is sent and accounted for independently of any buffered data messages and its delivery is usually prompt. Interrupt messages are limited in length to 16 bytes. They are most effectively used as event indicators and usually require the subsequent exchange of data by the two processes owning the logical link.

5.3.13 Sending Normal Data

To send a data message to another task, the program executes the Send Normal Data function (.NSFDS). The .NSFDS argument list has the format:

```
.NSAFN    FLAGS+XWD .NSFDS,4
.NSACH    XWD status, channel number
.NSAA1    EXP byte count
.NSAA2    Byte-pointer to data
```

The program must include a count of the number of bytes in the message and a byte pointer (.NSAA2) to the data in a program buffer. As DECnet-1Ø moves the data from the program buffer to the monitor buffers, it decrements the byte count and advances the byte pointer. DECnet-1Ø then sends the data to the remote node, segmenting it as necessary to comply with the segment size.

If the program sets the NS.EOM flag, the program buffer represents a message or the final portion of a message. This can force DECnet-1Ø to send the data even if it does not fill a segment. Note that the program must set the NS.EOM flag before executing the Synchronous Disconnect function or the disconnect function fails.

If the program sets the NS.WAI flag, it waits until the UUO returns. The UUO returns when DECnet-1Ø transmits the entire buffer of data. If the program does not set the NS.WAI flag, the UUO returns when the program's quota of monitor buffers is exhausted. If the entire buffer of data has not been sent, the byte count is non-zero. The program must check the byte count to determine whether all the data was sent, and execute the Send Normal Data function again if necessary.

If the link is in a state other than running (.NSSRN) or Connect Sent (.NSSCS), the UUO takes the error return. If the link is in Connect Sent state, the NS.WAI flag must be set so that the program can wait for the state to change to Running, otherwise the UUO takes the error return.

Example of Send Normal Data Function

```
SNDMSG.
        HRRM    CHAN,DSBLK+.NSACH;Store channel number into arg block
        MOVEI   T1,DSBLK        ; Point to Send Normal Data arg block
        NSP.    T1,             ; Send Normal Data
          JRST  [OUTSTR [ASCIZ /?Can't  Send Normal Data /]; Error
                 JRST    EREXIT] ; Return

; Argument block for Send Normal Data
DSBLK:  NS.WAI+ NS.EOM+XWD .NSFDS,4     ;Wait and end-of-message bits
                                ; Function, block length
        XWD     Ø,Ø             ; Code supplies channel number
        EXP     33              ; Number of bytes in message
        POINT   7,MESSAG        ; Byte pointer to message to be sent

; Message to be transmitted
MESSAG: ASCII   /HI! THIS IS HOSTA SPEAKING!/
```

5.3.14 Receiving Normal Data

To receive a data message from another task, the program executes the Receive Normal Data function (.NSFDR). The .NSFDR argument list has the format:

```
     .NSAFN     Flags+XWD  .NSFDR,4
     .NSACH     XWD status, channel number
     .NSAA1     EXP byte count
     .NSAA2     Byte pointer to data buffer
```

The program must include a count of the number of bytes expected and a byte pointer to the buffer that will hold the incoming data.

As DECnet-1Ø moves data from the monitor buffers to the program buffer, it decrements the byte count and advances the byte pointer. To determine the number of bytes received, the program must subtract the value of the byte count after execution from the value specified in the call.

The program will never receive data from more than one message in a single execution of this function. If the program does not set the NS.EOM flag, (or clears it from a previous call), DECnet-1Ø returns as much of the message as will fit into the buffer. If the program sets the NS.EOM flag, DECnet-1Ø returns as much of the message as will fill the buffer and discards the excess data. If DECnet-1Ø discards any data, it sets the byte count to the negative of the number of bytes discarded. Thus, a byte count of zero or greater means that the message fit into the buffer. If the program sets NS.EOM and does not set NS.WAI as well, the .NSFDR function will fail.

If the program sets the NS.WAI flag, the program waits until DECnet-10 transfers an entire message into the buffer or the buffer is full. If the program does not set the NS.WAI flag, the UUO returns immediately unless there is data waiting. If so, DECnet-10 returns either an entire message or as much of a message as will fill the buffer. The UUO takes the error return if the NS.EOM flag is set and the NS.WAI flag is not. This is to avoid the deadlock possible if the remote task were to send a message larger than the local task's monitor buffer quota.

If the link is in a state other than Running (.NSSRN) or Connect Sent (.NSSCS), the UUO takes the error return. If the link is in Connect Sent state, the NS.WAI flag must be set so that the program can wait for the state to change to Running, otherwise, the UUO takes the error return.

Example of the Receive Normal Data Function

```
READMS:
        HRRZM   CHAN,DRBLK+.NSACH;Store channel number into arg block
        MOVEI   T1,500          ; Maximum number of bytes
        MOVEM   T1,DRBLK+.NSAA1 ; Store into argument block
        MOVE    T1,[POINT 7,MESSAG]; Same with message byte pointer
        MOVEM   T1,DRBLK+.NSAA2
        MOVEI   T1,DRBLK ; Point to Receive Normal Data arg block
        NSP.    T1,             ; Receive Normal Data
          JRST  REDERR          ; Error in Receive Normal Data

; When control comes here, there's a message
        MOVE    T1,[POINT 7,MESSAG]; Get byte pointer to message
        MOVEI   T2,500          ; Get size of buffer in bytes
        SUB     T2,DRBLK+.NSAA1 ; Subtract number of bytes in buffer
                                ; to get number of bytes in message
        JUMPLE  T2,READM1       ; Skip output if no characters
; Output loop
        ILDB    T3,T1           ; Get first/next character
        OUTCHR  T3              ; Type out character
        SOJG    T2.-2           ; Count characters and loop

; If end-of-message bit is on, type <End of message> on terminal.
READM1: MOVE    T1,DRBLK+.NSAFN ; Get flag/function,length word
        TLZE    T1,(NS.EOM)     ; Test for EOM and turn off EOM Bit
        OUTSTR  [ASCIZ /        <End of message>/]

        MOVEM   T1,DRBLK+.NSAFN ; Turn off EOM bit in arg block for
                                ; next call
        JRST    READMS          ; Go read another message

; Argument block for Receive Normal Data
DBRLK:  NS.WAI+XWD  .NSFLR,4    ; Wait bit, function, block length
        XWD     0,0             ; Code supplies channel number
        EXP     500             ; Maximum number of bytes in message
        POINT   7,MESSAG        ; Byte pointer to message

; Message to be received
MESSAG: BLOCK   100             ; Space for 500 character message
```

5.3.15 Sending Interrupt Data

To send an interrupt message to another task, the program executes the Send Interrupt Data function (.NSFIS). The .NSFIS argument list has the format:

```
.NSAFN    Flags+XWD .NSFIS,3
.NSACH    XWD Status, Channel number
.NSAA1    Pointer to string block containing the data
```

The pointer in .NSAA1 points to a string block containing the data. The data cannot be longer than 16 bytes.

The program must set the byte count and block length in the string block. However, DECnet-1Ø ignores any bytes beyond the maximum of 16. DECnet-1Ø does not segment interrupt messages, even if the NS.WAI flag is not set. If there are outstanding interrupt data requests from the remote node, the Send Interrupt Data function causes DECnet-1Ø to send the interrupt data.

The UUO takes the error return under any of the following conditions:

- There are no outstanding interrupt data requests
- There is already an interrupt message for transmission
- The link is not in Running or Connect Sent state
- The NS.WAI flag is not set while the link is in the Connect Sent state

5.3.16 Receiving Interrupt Data

To receive an interrupt message from another task, the program executes the Receive Interrupt Data function (.NSFIR). The .NSFIR argument list has the format:

```
.NSAFN    Flags+XWD .NSFIR,3
.NSACH    XWD Status, Channel number
.NSAA1    Pointer to a string block to receive data
```

.NSAA1 contains a pointer to a string block that will contain the data. The maximum block size is 16 bytes. Interrupt messages cannot exceed that size. If the string block is too small, DECnet-1Ø truncates the message.

The program must set the byte count and block length in the string block, but DECnet-1Ø ignores any space not necessary to hold data. DECnet-1Ø either receives the whole message or none of it.

The interrupt message does not, of itself, cause an interrupt. It can bypass queued normal data messages. The program must enable the PSI system and set the PSI reason mask appropriately to cause an interrupt.

If the link is in a state other than Running (.NSSRN) or Connect Sent (.NSSCS) the UUO takes the error return. If the link is in Connect Sent state, the NS.WAI flag must be set so that the program can wait for the state to change to Running. Otherwise, the UUO takes the error return.

5.3.17 Closing a Network Connection

Either of the two connected tasks can close a network connection. A connection can be closed normally, thereby preserving the integrity of any data in transit, or a connection can be aborted without regard to any undelivered data.

To close a connection normally, the program executes the Synchronous Disconnection function (.NSFSD). (The Synchronous Disconnect function may be used by synchronously and asynchronously running programs.) The .NSFSD argument list has the format:

```
.NSAFN    XWD .NSFSD, length (length = 2 or 3)
.NSACH    XWD Status, Channel number
.NSAA1    String block pointer to disconnect data (optional)
```

.NSAA1 contains a pointer to an optional string block containing disconnect data. This data cannot be longer than 16 bytes.

If the Synchronous Disconnect function is executed successfully, DECnet-10 sends a disconnect initiate message to the remote task and puts the link into Disconnect Sent state (.NSSDS). While the link is in this state, the program cannot send any data, but should be prepared to read any data sent by the other task before it received the disconnect. When the remote task confirms the disconnect, DECnet-10 places the link in Disconnect Confirmed state (.NSSDC). The program can then release the channel. If the program releases the channel (using .NSFRL, RESET. or EXIT) without waiting for the disconnect message, unacknowledged messages from the remote task could be lost.

To insure that messages are not lost, the program should not set the NS.WAI flag and should be prepared to read any further messages. If the PSI system is enabled and the reason mask is set for data available, an interrupt occurs when a message arrives. If the program has also set the reason mask for disconnect confirm, the arrival of a disconnect confirm message also causes an interrupt. If not enabled for interrupts, the program can read any left-over messages by executing one of the Receive Data functions (see the .NSFDR and .NSFIR functions) with the NS.WAI flag set. By doing so, the program also determines the arrival of the disconnect confirm message because either function takes the error return for a disconnect confirm message.

If the link is in any state other than Running (.NSSRN) or if the last message sent did not have the end-of-message (NS.EOM) flag set, the UUO takes the error return.

Example of the Synchronous Disconnect Function

```
SYNDSC:
        HRRM    CHAN,SDBLK+.NSACH; Store channel number into arg block
        MOVEI   T1,SDBLK          ; Point to Sync Disconnect arg block
        NSP.    T1,               ; Synchronous Disconnect
          JRST  [OUTSTR [ASCIZ /?Can't Sync Disconnect /]; Error
                JRST    EREXIT]   ; Return

; Argument block for Synchronous Disconnect
SDBLK:  NS.WAI+XWD .NSFSD,2       ; Wait bit, function, block length
        XWD     0,0               ; Code supplies channel number
```

5.3.18 Releasing a Channel

After receiving a disconnect received or disconnect confirm message, the task can execute the Release Channel function (.NSFRL) to release the channel. The .NSFRL argument list has the format:

```
.NSAFN    XWD .NSFRL,2
.NSACH    XWD status, channel number
```

The UUO returns immediately, even if the NS.WAI flag is set.

When a program receives a disconnect initiate message (.NSSDR state), it executes the Release Channel function to confirm the disconnect, release the link, and return all buffers for that link. DECnet-1Ø sends a message to the other task confirming the disconnect.

When a program receives a disconnect confirm message, it executes the Release Channel function to release the link and return all buffers used for that link.

A program that has executed the Enter Passive function and is waiting for a connection (.NSSCW state) can execute the Release Channel function to release the link and any buffers allocated for the link. Since no connection has yet been made, DECnet-1Ø does not send a disconnect confirm message.

If the link is in any state other than Connect Wait (.NSSCW), Disconnect Received (.NSSDR), or Disconnect Confirmed (.NSSDC), the link is immediately released without any disconnect message sent to the other task. This is an abrupt aborting of the link.

Example of the Release Channel Function

```
RELCHN:
        HRRM    CHAN,RLBLK+.NSACH; Store channel number into arg block
        MOVEI   T1,RLBLK          ; Point to Release Channel arg block
        NSP.    T1,               ; Release Channel
          JRST  [OUTSTR [ASCIZ /?Can't Release Channel /]; Error
                 JRST    EREXIT]  ; Return

; Argument block for Release Channel
RLBLK:  NS.WAI+XWD .NSFRL,2       ; Wait bit, function, block length
        XWD     Ø,Ø               ; Code supplies channel number
```

5.3.19 Aborting a Connection

To abort the connection, release the link, and send an abort message, the program executes the Abort and Release function (.NSFAB). The .NSFAB argument list has the format:

```
.NSAFN    XWD .NSFAB,length (length = 2 or 3)
.NSACH    XWD status, channel number
.NSAA1    Pointer to disconnect data (optional)
```

.NSAA1 can contain a pointer to an optional string block containing disconnect data. This data cannot be longer than 16 bytes.

If the link is in Running state (.NSSRN) and the Abort and Release function is executed successfully, DECnet-1Ø sends an abort message to the other task and immediately releases the link and all buffers allocated to that link. The other task should release the link on its end to complete the disconnect. If the link is in a state other than Running, the UUO takes the error return.

5.3.20 Reading the Disconnect Data

As a result of a Synchronous Disconnect, an Abort and Release, or a Reject Connect function, DECnet-10 sends a disconnect message to the other task. To read the data from the disconnect message, the program executes the Read Disconnect Data function (.NSFRD). The .NSFRD argument list has the format:

```
.NSAFN    XWD .NSFRD,length (length = 3 or 4)
.NSACH    XWD status, channel number
.NSAA1    Pointer to string block to receive the data
.NSAA2    Reason code (optional)
```

.NSAA1 contains a pointer to a string block that will receive any data that the other task included with its disconnect function. If there is no data, DECnet-10 leaves the string block empty. The data cannot be longer than 16 bytes.

DECnet-10 sets .NSAA2 to a reason code that specifies the reason for the disconnect. These reason codes are not from the other task, but from DECnet-10 and are universal to all DECnet implementations. The possible reason codes and their meanings are:

```
 0 - Rejected or disconnected by object (task)
 1 - No Resources
 2 - Unrecognized node name
 3 - Remote node shut down
 4 - Unrecognized object
 5 - Invalid object name format
 6 - Object too busy
 8 - Abort by network management
 9 - Abort by object
10 - Invalid node name format
11 - Local node shut down
34 - Access control rejection
38 - No response from object
39 - Node unreachable
41 - No link or logical link has been lost
42 - Disconnect complete
43 - Image field too long (rqstrid, password, account, usrdata, etc.)
44 - Unspecified reject reason
```

Note that some of these reasons apply to a task that has rejected a connection. DECnet-10 sends a disconnect initiate message when a task rejects a connection as well as when the task disconnects the link.

If the link is in Disconnect Received state, (.NSSDR) the function is executed successfully and the link remains in that state. If the link is in any state other than Disconnect Received, the UUO takes the error return.

5.4 OBTAINING INFORMATION ABOUT DECNET-1Ø

For monitors that support DECnet-1Ø Version 3 and Version 4 networks, the DNET. monitor call allows you to obtain data about the nodes in the DECnet area you are in, and the DECnet links used for intertask communication. The DNET. call has three functions:

Fcn-code	Symbol	Meaning
1	.DNLNN	Lists the names of the nodes in the network.
2	.DNNDI	Returns information about each node in the network as it relates to the EXECUTOR node (that is, the node at which your program is running).
3	.DNSLS	Returns information about each logical link established from your node. A logical link is the path of communication used for DECnet intertask communication.

Function 1 for DNET. is .DNLNN, to list the node names of the nodes in the DECnet area. The calling sequence and argument block that you must supply for .DNLNN is:

```
        MOVEI    ac,addr
        DNET.    ac,
          error return
        normal return

addr:   flags+.DNLNN,,length+1
        BLOCK length
```

where **addr** points to the argument list. **Length** specifies the number of words to reserve in the argument block for the returned list of node names. The maximum number of nodes in a DECnet-1Ø area is 1Ø24, so the value of **length** should not exceed this. The symbol .DNLLN is defined to have this value. If the length is too short for the list of returned node names, the list will be truncated to the number of words reserved.

You must set one of the following flags for .DNLNN:

Bits	Symbol	Meaning
1	DN.FLK	List only "known" nodes. That is, the list of node names returned will be all the nodes that are defined with node names plus all nodes that are reachable. If the system is running as an Ethernet end node, the only node it knows is itself. The ST%END entry in the %CNST2 GETTAB table indicates whether the DECnet is running as an Ethernet endnode.
2	DN.FLR	List only reachable nodes. This flag restricts the returned list to only those nodes that are currently in the network.
3	DN.FLE	List only the EXECUTOR node. This is the node at which the program is running.

When the information is returned at addr+1, the form of the returned block is:

```
addr:   flags+.DNLNN,,length
        count
        first node name
        second node name
          .
          .
          .
```

where Word Ø (.DNFFL) contains the same information you place in this word. Word 1 (.DNCNT) contains the number of node names returned. Words 2 and following (.DNNMS) each contain a SIXBIT node name.

Example

The following is an example of the use of .DNLNN:

```
        MOVE    T1,[<.DNLNN>B17!DN.FLK!.DNLLN]  ;Function word to return
                                                 ; known nodes up to the
                                                 ; maximum length arg block
        (.DNLLN)
        MOVEM   T1,DNARG                         ;Save word in arg block
        MOVEI   T1,DNARG                         ;Point to arg block
        DNET.   T1,                              ;Ask for information
          HALT                                   ;Shouldn't happen
```

At this point, the argument block should be:

```
DNARG:  DN.FLK!<.DNLNN>B17!.DNLLN
        2Ø                                       ;2Ø nodes
        SIXBIT/ONE/                              ;First node
        SIXBIT/TWO/                              ;Second node
        SIXBIT/THREE/
        SIXBIT/KL1Ø26/
        SIXBIT/JINX/
        SIXBIT/GNOME/
          .
          .
          .
```

Function 2 of DNET. is .DNNDI, which returns information about a specific node in the network or about all the nodes in the network. There are two methods of using this function: step control and non-step control. In step mode, the information is returned for each node in the network, arranged by node address. Under step mode, you must clear addr+1 on the first call, so that the information returned will begin with the first node in the node list. If you do not set the step mode flag, information is returned only for the node whose name you specify in addr+1. The control method is set by flag DN.FLS. Specifically, the calling sequence and argument block for .DNNDI is:

```
        MOVE    ac,addr
        DNET.   ac,
          error return
        normal return

addr:   flags+.DNNDI,,length
        node name
        BLOCK n
```

where the flags are:

Bits	Symbol	Meaning
Ø	DN.FLS	Set step mode for controlling returned information about every node in the DECnet network.
1	DN.FLK	List information about known nodes. Set this flag only if you set step mode (DN.FLS).
2	DN.FLR	List reachable nodes. Set this flag only if you set step mode (DN.FLS).
3	DN.FLE	List EXECUTOR node.

If you do not set DN.FLS, you can return information only about the node whose name you specify, in SIXBIT, in addr+1. The information is returned in the argument block as follows:

```
addr:   flags+.DNNDI,,length
        node-name
        router information
        link information
        node address
        circuit name
```

where each word at addr is returned as follows:

Word	Symbol	Contents
Ø	.DNFFL	Flag word that you specified.
1	.DNNAM	The node name of the currently listed node.
2	.DNRTR	Router information. Bit Ø (DN.RCH) of this word is set if the node is reachable. If bit Ø equals Ø, the node is not reachable. The remainder of the left half of this word (DN.HOP) contains the number of hops (node-to-node paths) over which a signal must pass to get from the EXECUTOR node to the node in addr+1. The right half of this word (DN.CST) contains the relative cost of the path to the specified node.
3	.DNLLI	Link information. Bit Ø (DN.VLD) is set if the rest of this word contains valid information. In this case, the left half of the word indicates the number of logical links currently established between the local and remote nodes. If bit Ø is off, the information in this word is not valid, indicating that no attempt has been made to communicate with the remote node.
4	.DNADR	Node address.
5-1Ø	.DNCKT	Circuit information. This information describes the controller of the line that is the first hop in the path to the remote node. The circuit name is up to 4 ASCIZ words.

Example

The following example shows how the DNET. function .DNNDI might be used in step mode to show information about known nodes:

```
MOVE    T1,[<.DNNDI>B17!DN.FLK!DN.FLS!.DNNLN] ;Show link status
                                               ; up to the maximum
                                               ; length of the
                                               ; argument block (.DNNLN)
MOVEM   T1,DNARG                               ;Save in arg block
SETZM   DNARG+.DNNAM                           ;Clear arg+1 to start at
                                               ; First node name
MOVEI   T1,DNARG                               ;Point to arg block
DNET.   T1,                                    ;Ask for information
  HALT                                         ;Shouldn't happen
```

The argument block returned may be:

```
DNARG:  DN.FLK!DN.FLS!<.DNNDI>B17!.DNNLN ;Flag word
        SIXBIT/ONE/                      ;Node name
        1B0!3B17!4B35                    ;Reachable,3 hops, cost=4
        0                                ;No active links
        1                                ;Node address=1
        ASCII/DTE-0/                     ;Circuit-id
        ASCII/-1/                        ;Continuation of
                                         ; Circuit-id
```

Function 3 of DNET., .DNSLS, lists information about DECnet logical links. Every intertask communication through DECnet is performed over a logical link (also known as a DECnet channel). Refer to Section 5.3 for information about establishing DECnet links. The calling sequence and argument list for .DNSLS is:

```
        MOVE    ac,addr
        DNET.   ac,
          error return
        normal return

addr:   DN.FLS+<.DNSLS,,length>
        jobno,,channo
        BLOCK n
```

where Bit 0 (DN.FLS) of the word at **addr** is optional. If set, DN.FLS sets step control for the information returned. Under .DNSLS, step mode returns information about each link that is open from the EXECUTOR node, in the order of job number, starting with job 1, and by link number within the job. After all the jobs are listed, then job number -1 is listed. Under step mode, you should clear addr+1 (.DNJCN) the first time the call is issued.

If DN.FLS is not set, non-step mode is the control method, and addr+1 (.DNJCN) should contain the job number and DECnet channel number of the link for which you want the following information.

The link information is returned to you in the following form:

Word	Symbol	Contents
Ø	.DNFFL	The flag word you specified in the argument block.
1	.DNJCN	The job number and channel number for which the following information is appropriate.
2	.DNNOD	The node name (in SIXBIT) of the node to which the link is made.
3	.DNOBJ	The object types of the communicating jobs. The standard object types are listed in UUOSYM.MAC. The left half of this word (DN.DOB) contains the object type of the destination process, and the right half (DN.SOB) contains the object type for the source process. If a non-zero format type is used for transmission, this entire word is Ø.
4	.DNSTA	Contains the status of the link. The left half (DN.LSW) contains the binary representation of the link status (refer to Volume 2). The right half (DN.STA) contains the two-letter status of the link, in SIXBIT. These status symbols are:

Symbol	Meaning
CF	No confidence
CM	No communication
CR	Connect received
CS	Connect sent
CW	Connect wait
DC	Disconnect confirmed
DR	Disconnect received
DS	Disconnect sent
LK	No link
NR	No resources
RJ	Remote node rejected connect initiate message
RN	Link is running

5	.DNQUO	Quota word, where the left half (DN.QUI) contains the input quota, and the right half (DN.QUO) contains the output quota of outstanding messages allowed on the link.
6	.DNSEG	Contains the segment size.
7	.DNFLO	Contains the flow control option, where the left half (DN.XMF) is the flow control for transmission, and the right half (DN.RCF) is the flow control for reception of messages.

Word	Symbol	Contents
10	.DNMSG	Contains the message count, where the left half (DN.MRC) contains the number of messages received, and the right half (DN.MXM) contains the number of messages sent over the link.
11	.DNMPR	Contains the monitor process or the terminal number of the job associated with the channel number in .DNJCN. If the job number is -1, this words contains the TTY number corresponding to the monitor process NRTSER's link.

Example

The following example shows how the .DNSLS function might be used to obtain information about the first link for job 1, under step control. .DNSLN is the maximum block length for the function.

```
MOVE    T1,[<.DNSLS>B17!DN.FLS!.DNSLN]  ;Link status, step
                                         ; control
MOVEM   T1,DNARG                         ;Save in arg block
SETZM   DNARG+.DNJCN                     ;Start at first link
MOVEI   T1,DNARG                         ;Point to arg block
DNET.   T1,                              ;Ask for information
  HALT                                   ;Shouldn't happen
```

At this point, the argument block should look like:

```
DNARG:  DN.FLS!,.DNSLS>B17!.DNSLN        ;Flag word
        -1,,1                            ;NRTSER pseudo-job,
                                         ; channel 1
        SIXBIT/THREE/                    ;Connected to node THREE
        27,,27                           ;NRT object type on both
                                         ; sides of the link
        240005,,'RN'                     ;Binary status,running
                                         ; link
        10,,10                           ;10 credits either
                                         ; direction
        100                              ;Segment size
        2,,1                             ;Segment flow control,,no
                                         ; flow control
        2073,,1241                       ;Messages sent and
                                         ; received
        110                              ;Link is NRT TTY110
```

5.5 ETHERNET NETWORKS

The ETHNT. monitor call allows you to develop custom protocols for the Ethernet hardware. Writing software using the function codes described in Volume 2 gives you access to any device or system that is connected to the Ethernet. Once your program accesses the device or system, data may be transmitted and received between the devices or systems.

ETHNT. uses buffers called datagrams for information exchange over the Ethernet. Datagrams are transmitted on logical communication channels called portals. Portals uniquely identify particular Ethernet users, by the portal ID. Executing the Open User Portal function generates a portal ID, which is a 27-bit long, randomly assigned number. Information associated with each portal includes a protocol type and (optionally) one or more multicast addresses. A multicast address is an address meant for multiple destinations on the same Ethernet.

A protocol type indicates to the Ethernet the type of network communications associated with the portal. Decimal values from Ø to 65535 are interpreted as protocol types. Each protocol type must not be associated with any other existing portals in the system. If the type contains -1, then the portal has no protocol type associated with it. If the type contains -2, promiscuous mode is enabled. If the protocol type contains -3, the Ethernet assigns the value "Unknown Protocol Type Queue" to the portal. This queue receives messages that do not match any enabled protocol type. Types -2 and -3 are not implemented.

5.5.1 Transmitting and Receiving Information

When you queue a receive or transmit buffer (ETHNT. functions .ETQRB and .ETQXB, respectively), you must include a user buffer descriptor list (also called a descriptor block) that contains information about the buffer. The .ETUBL argument for these functions contains the address of this buffer descriptor list. When you read the transmit or receive queues (ETHNT. functions .ETRRQ and .ETRXQ, respectively), you must also include the address of the buffer descriptor list.

Each of these descriptor blocks contains a status word, .UBSTS. This word indicates when a buffer has been transmitted or received successfully. If a failure occurred, the word indicates the nature of the failure. You should examine this word upon the completion of a transmit or receive.

The format of the User Buffer Descriptor Block is:

Word	Symbol	Contents
Ø	.UBNXT	Address of the next user buffer descriptor.
1	.UBBID	User buffer ID.
2	.UBSTS	User buffer status. This may return with the flag UB.ERR, indicating an error occurred in transmitting or receiving the buffer. UB.ECD in this word contains the error code if the error flag is set.
3	.UBBSZ	Length of the datagram (in bytes).
4	.UBBFA	A byte pointer to the datagram.
6	.UBPTY	Protocol type.
7	.UBDEA	A two-word Ethernet destination address.
11	.UBSEA	A two-word Ethernet source address.

The User Buffer descriptor block has a length of 13; .UBLEN is the symbol for block length.

An example of the ETHNT. queue receive buffers function follows. This function includes returns from .ETRRQ, the Read Receive Queue function.

```
            XMOVEI    ac,addr
            ETHNT.    ac
              error return
            normal return

addr:       .ETQRB,,3                   ;Function,,block length
            portal ID                   ;Portal ID
            UBL                         ;Address of buffer
                                        ; descriptor list

UBL:        Ø                           ;Address of next buffer
                                        ; descriptor list
            buffer ID                   ;Returned on .ETRRQ
                                        ; function
            Ø                           ;Status, returned on
                                        ; .ETRRQ
            1ØØØ                        ;Datagram length in bytes
            POINT 8,DGM                 ;Byte pointer
            Ø
            Ø                           ;Protocol type, returned
                                        ; on .ETRRQ
            Ø                           ;Destination Ethernet
            Ø                           ; addr,returned on .ETRRQ
            Ø                           ;Source Ethernet addr,
            Ø                           ; returned on .ETRRQ

DGM:        BLOCK <1ØØØ+3>/4            ;Space for datagram
```

5.5.2 Returned Channel Information

When you request the Read Channel Information function (.ETRCI) or the Read Channel Counters function (.ETRCC), ETHNT. returns a buffer that contains the requested information. The buffers are different for each function. The returned information will be at the address you specified in the .ETBFA word of the function. You specify the buffer length (in words) in the .ETBFL word of the function. (Refer to Volume 2 for a description of these words.)

The form of the return buffer for .ETRCI is:

Word	Symbol	Contents
Ø	.EICNM	The Ethernet channel number.
1	.EICEA	The current (two-word) Ethernet address.

The form of the return buffer for .ETRCC is:

Word	Symbol	Contents
Ø	.ECCSZ	Seconds since the counters were last zeroed.
1	.ECCBR	Number of bytes received.
2	.ECCBX	Number of bytes transmitted.
3	.ECCDR	Datagrams received.
4	.ECCDX	Datagrams transmitted.
5	.ECCMB	Multicast bytes received.
6	.ECCMD	Multicast datagrams received.
7	.ECCXD	Datagrams transmitted, but initially deferred.
1Ø	.ECCX1	Datagrams transmitted, single collision. A collision occurs when two station's transmissions overlap.
11	.ECCXM	Datagrams transmitted, multiple collisions.
12	.ECCXF	Transmit failures.
13	.ECCXX	The transmit failures bit mask. The bits are:

Bit	Symbol	Meaning
28	EC.XCL	Carrier was lost.
29	EC.XBP	Transmit buffer parity error.
3Ø	EC.XFD	Remote failure to defer.
31	EC.XFL	Transmitted frame too long.
32	EC.XOC	Open circuit.
33	EC.XSC	Short circuit.
34	EC.XCC	Collision detect check failed.
35	EC.XEC	Excessive collisions.

Word	Symbol	Contents
14	.ECCRF	Receive failures.
15	.ECCRX	The receive failures bit mask. The bits are:

Bit	Symbol	Meaning
31	EC.RFP	Free list parity error.
32	EC.RNB	No free buffers.
33	EC.RFL	Frame too long.
34	EC.RFE	Framing error.
35	EC.RBC	Block check error.

Word	Symbol	Contents
16	.ECCUD	Unrecognized frame destination.
17	.ECCDO	Data overrun.
2Ø	.ECCSU	System buffer unavailable.
21	.ECCUU	User datagram buffer unavailable.

5.5.3 Returned Portal Information

When you request the Read Portal Information function (.ETRPI) or the Read Portal Counters function (.ETRPC), ETHNT. returns a buffer containing the information. The buffers are different for each function. The returned information will be at the address you specified in the .ETBFA word of the function you requested. You specify the buffer length (in words) in the .ETBFL word of that function as well.

The format of the returned Read Portal Information buffer is:

Word	Symbol	Meaning
Ø	.EIPCJ	Job Context Handle (JCH) of the portal owner.
1	.EIPPI	Protocol identification word.
2	.EIPCS	Channel status word.
3	.EIPKC	Controller status word.

The returned buffer for a Read Portal Counters has the format:

Word	Symbol	Meaning
Ø	.ECPSZ	Seconds since the counters were last zeroed.
1	.ECPBR	Bytes received.
2	.ECPDR	Datagrams received.
3	.ECPBX	Bytes transmitted.
4	.ECPDX	Datagrams transmitted.
5	.ECPUU	User datagram buffer unavailable.

5.5.4 Returned Controller Information

When you request the Read Controller Information function (.ETRKI) or the Read Controller Counters function (.ETRKC), ETHNT. returns a buffer that contains the requested information. The buffers are different for each function. The returned information will be at the address you specified in the .ETBFA word of the function. You specify the buffer length (in words) in the .ETBFL word of the function.

The buffer returned for a Read Controller Information request is:

Word	Symbol	Meaning
Ø	.EIKCS	Channel status word.
1	.EIKCP	CPU number of controller.
2	.EIKTY	Controller type. A value of 1 (EI.KNI) is returned for an NIA2Ø interface.
3	.EIKNO	Controller number.
4	.EIKHA	A two-word Ethernet hardware address.

The buffer returned for a Read Controller Counters request is:

Word	Symbol	Contents
Ø	.ECKSZ	Seconds since the counters were last zeroed.
1	.ECKBR	Number of bytes received.
2	.ECKBX	Number of bytes transmitted.
3	.ECKDR	Datagrams received.
4	.ECKDX	Datagrams transmitted.
5	.ECKMB	Multicast bytes received.
6	.ECKMD	Multicast datagrams received.
7	.ECKXD	Datagrams transmitted, but initially deferred.
1Ø	.ECKX1	Datagrams transmitted, single collision.
11	.ECKXM	Datagrams transmitted, multiple collisions.
12	.ECKXF	Transmit failures.
13	.ECKXX	The transmit failures bit mask. The bits are listed in Section 5.5.2

Word	Symbol	Contents
14	.ECKRF	Receive failures.
15	.ECKRX	The receive failures bit mask. The bits are listed in Section 5.5.2
16	.ECKUD	Unrecognized frame destination.
17	.ECKDO	Data overrun.
20	.ECKSU	System buffer unavailable.
21	.ECKUU	User datagram buffer unavailable.

CHAPTER 6

TRAPPING, INTERCEPTING, AND INTERRUPTING

Assembly language programs can handle errors and interrupt execution when specific conditions occur, using the following methods:

- Trapping an error and jumping to a trap-service routine. Traps can be set for several kinds of errors, including illegal memory references, pushdown list overflows, and arithmetic overflows (generally CPU conditions).

- Intercepting specific kinds of errors. The monitor then transfers control to a predefined address for error handling (generally I/O conditions).

- Using the software interrupt facility. The program can define any or all of a wide variety of interrupt conditions; when one of these conditions occurs, the monitor transfers control to a predefined address for interrupt handling.

Traps are synchronous events that often occur in the normal execution of a program (for example, a context switch). After a trap, the PC is predictable. A UUO, for example, causes a trap to the monitor. **Interrupts**, however, are asynchronous conditions caused by external events, interrupting program execution because of a specific change in a condition or word. After an interrupt, the PC is usually not predictable (for example, if the interrupt occurs during I/O that takes multiple instructions, the program can only detect the interrupt after I/O is completed).

This chapter describes methods for handling halts, errors, and software conditions in a TOPS-10 assembly language program. Throughout this chapter, the term "interrupt" refers to a condition produced by the software interrupt facility as opposed to the CPU's hardware interrupt system. Using the software interrupt system is more versatile than trapping and intercepting errors.

NOTE

APRNEB traps and the .JBINT intercept block are supported for Section 0 programs only. Programs that require error handling and interrupting in non-zero sections must use the Programmed Software Interrupt (PSI) system.

6.1 TRAPPING ERRORS AND CONDITIONS

Your program can trap errors by using the APRENB monitor call to enable certain kinds of traps and then handle the traps that occur with special trap-servicing routines.

To set an APRENB trap, your program must:

1. Place the address of the trap-service routine in location .JBAPR in the job data area.

2. Enable one or more conditions for trapping by issuing an APRENB monitor call.

3. Include a trap service routine at the address given in .JBAPR. This routine should test to see which condition occurred, and (if the program is to regain control) should end with the following instruction:

   ```
   JRSTF @.JBTPC
   ```

   This instruction clears the bits that indicated the trap condition, restores the state of the processor, and continues the program.

APRENB traps can be set to handle the following conditions. The address specified in the error messages is your current PC.

- Pushdown list overflows. If your program does not trap these overflows, the monitor stops the job and prints:

  ```
  ?Pushdown list overflow at user PC address
  ```

- Illegal memory references. If your program does not trap these violations, the monitor stops your job and prints:

  ```
  ?Illegal memory reference at user PC address
  ```

  Or, if the reference is from your program's PC:

  ```
  ?PC out of bounds at user PC address
  ```

- References to nonexistent memory. If your program does not trap these references, the monitor stops your job and prints

  ```
  ?Non-existent memory at user PC address
  ```

- Memory parity errors. If your program does not trap these errors, the monitor stops the job and prints:

  ```
  ?Memory parity error at user PC address
  ```

- Clock ticks while the program is running. These conditions are not errors, and the monitor does not expect your program to trap them.

- Floating-point overflows. If your program does not trap these overflows, the monitor ignores the condition and continues your job. A condition such as this can cause a hazardous situation for your program that could result in fatal errors later in execution.

- Arithmetic overflows. If your program does not trap these overflows, the monitor ignores the condition and continues your job. A condition such as this can cause a hazardous situation for your program that could result in fatal errors later in execution. The UTRP. call is the preferred method for handling arithmetic overflows.

When an enabled trap condition occurs, the monitor performs the following functions:

1. Leaves a bit mask of conditions in .JBCNI in the job data area. The bit mask is the same as the APRENB bits AP.xxx.

2. Moves the current program counter (PC) to the location .JBTPC in the job data area, clearing the floating point and arithmetic overflow flags. Note that the PC at this time is pointing to the next instruction to be executed and not the instruction that caused the trap. If this PC points to the first or second instruction in the trap service routine, the monitor stops your job.

3. Transfers control to the address given in location .JBAPR in the job data area. This is the address you have set as the beginning of the trap service routine.

In general, each time a trap occurs the monitor clears the trap conditions that your program set. To enable repetitive trapping, you must set the AP.REN bit when executing the APRENB monitor call. This repetitive-enable does not affect clock-tick traps. To trap consecutive clock ticks, your program must explicitly re-enable the trap condition.

6.2 INTERCEPTING ERRORS

For specific kinds of errors, the monitor intercepts control of your program automatically. Your program can allow the monitor to take some default action for these intercepts, or it can handle the intercepts with a special intercept block. This block allows you to accept or suppress the monitor's error message for the encountered error condition.

The errors that the monitor intercepts in this way are:

- Fatal errors in your job's execution. If these errors are not intercepted, they abort your job; the job cannot be continued. The possible error messages are:

  ```
  ?Illegal UUO at user PC address
  ?Address check at user PC address
  ?PC out of bounds at user PC address
  ```

  You can add APRENB error traps to your program or enable the software interrupt system and re-execute the program to determine the exact cause of the error.

- Jobs running past their time limits (nonbatch jobs only). This occurs when a job runs longer than the limit set by the SET TIME command. The error message is:

  ?Time limit exceeded

  You can issue the SET TIME command again, altering your job's time limit, and then use the CONTINUE or START monitor command to restart your program.

- Requests for space that would exceed your disk quota. The error message is:

  [Exceeding quota on str]

  Where str is the name of the relevant file structure. You must delete some files from your disk area before the program can execute properly.

- Requests for space on a file structure that is full. Your job must wait until some blocks are freed on the structure.

- CTRL/Cs typed from the controlling terminal. The monitor normally stops your program and puts your terminal in monitor mode. No message is displayed.

- Device errors that the operator can correct, such as an offline disk. The error message is:

  Device dev OPRnn Action requested

  Where dev is the name of the relevant device; and nn is the number of the node where the operator must take action.

  The operator at that node receives a message stating that there is a problem on the device. After fixing the device, the operator can continue your job by typing JCONTINUE. This generates the following message to your job:

  Cont by opr

  The function of JCONTINUE is automatically performed by the monitor once a minute.

The bits that affect the monitor's handling of these conditions are described in Table 6-1.

6.2.1 Using the .JBINT Intercept Block

To intercept these conditions, your program must:

1. Place the address of an intercept block in location .JBINT in the job data area.

2. Contain an intercept block at the address placed in .JBINT.

As with APRENB traps, .JBINT traps should resume the interrupted program (if desired) by using a JRSTF, where the return address is the interrupted PC as stored by the monitor in the trap block at offset .EROPC. Again, as with APRENB, the stored PC may or may not have anything to do with the intercept condition.

The intercept block contains the address to which control is to be transferred to handle the intercept, a message control bit, and bits specifying which errors are to be intercepted. The format of the intercept block is given in Table 6-1.

Table 6-1: Format of .JBINT Intercept Block

Word	Symbol	Contents
Ø	.ERNPC	Length and new PC, where the left half of this word is the length of the block (at least 3 words), and the right half is the new PC for the intercept-handling routine.
1	.ERCLS	Intercept message control and class flags. Bit Ø (ER.MSG) suppresses the error message that the monitor displays by default. The class flags are: Flag Symbol Error 29 ER.EIJ Error in job. 3Ø ER.TLX Time limit exceeded. 31 ER.QEX Disk quota exceeded. 32 ER.FUL File structure full. 33 ER.OFL Disk unit off line. 34 ER.ICC CTRL/C typed. 35 ER.IDV Problem with device.
2	.EROPC	Old PC. This location must be zero in the intercept block, unless you want the monitor to stop your job. The monitor fills this word with the interrupt PC when an intercept occurs.
3	.ERCCL	When an intercept occurs, the monitor fills this word with the channel number in the right half and the class of error that occurred in the left half. The class flags are listed under .ERCLS, with 18 added to the bit position.

For each type of error condition, there is an associated class flag. The monitor examines the class flags in .ERCLS and .EROPC to determine whether to interrupt your program or to stop the job. The monitor interrupts your program for an error if your program sets the appropriate flag in .ERCLS, and .EROPC is zero. It stops your job if you clear the appropriate flag in .ERCLS or if .EROPC is non-zero.

When the monitor interrupts your program, it returns the interrupted PC in .EROPC and the reason for the intercept in .ERCCL. Then it restarts your job at the location specified in .ERNPC.

6.2.2 Examples of Error Intercepts

The following example intercepts CTRL/Cs. The user is returned to the monitor level as quickly as possible.

```
        LOC     .JBINT              ;Set pointer to .JBINT
        EXP     INTBLK              ;Address of intercept block
        RELOC                       ;Resume relocatable

INTBLK: XWD     4,INTLOC            ;4 words long,,handler address
        EXP     ER.ICC              ;Intercept CTRL/Cs
        BLOCK   2                   ;For returned data

;Intercept routine starts here.

INTLOC: MOVEM   AC1,TEMP##          ;Save AC1
        MOVSI   AC1,ER.ICC          ;Get CTRL/C bit
        TDNN    AC1,INTBLK+.ERCCL   ;Was this a CTRL/C?
        HALT                        ;No

;Here if it was a CTRL/C.
;At this point the program should release any special resources,
; taking care to be quick and not to loop.

        . . .
        MONRT.                      ;Return to monitor

;Here if user CONTINUEs.

        MOVE    AC1,INTBLK+.EROPC   ;Get continuation PC
        EXCH    AC1,TEMP            ;Restore AC1
        SETZM   INTBLK+.EROPC       ;Clear to allow another intercept
        JRSTF   @TEMP               ;Resume where program stopped
```

The following example shows how to enable and handle a CTRL/C intercept, preventing the user from returning to monitor mode:

```
        LOC     .JBINT              ;Set pointer to .JBINT
        EXP     INTBLK              ;Address of intercept block
        RELOC                       ;Resume relocatable

INTBLK: XWD     4,INTLOC            ;4 words long,,handler address
        EXP     ER.ICC              ;Intercept CTRL/Cs
        BLOCK   2                   ;For returned data

;Intercept routine starts here.

INTLOC: SKIPN   CCEXIT              ;Can program be interrupted by
                                    ; CTRL/C?
        JRST    EXITOK              ;Yes, go clean up and exit
        SETOM   CCSEEN              ;Set flag that we saw a CTRL/C
        PUSH    P,INTBLK+.EROPC     ;Put interrupt PC in stack
        SETZM   INTBLK+.EROPC       ;Reenable intercept
        POPJ    P,                  ;Return to interrupted PC

CCEXIT: EXP     -1                  ;Flag set non-zero if a CTRL/C
                                    ;May not interrupt execution

CCSEEN: EXP     Ø                   ;Flag set non-zero by INTLOC
                                    ; if a CTRL/C was typed and
                                    ; CCEXIT was non-zero.

EXITOK: SETZM   INTBLK+.EROPC       ;Here if it was OK to interrupt
                                    ; execution of the program. Do
                                    ; any cleanup and exit.
```

6.3 USING PROGRAMMED SOFTWARE INTERRUPTS

Your job can use software interrupts that are generated by a wide variety of conditions. These interrupts allow your program to respond to external conditions and to requests for error servicing.

Most interrupts occur after the execution of one instruction and before the execution of the next. (It is possible for certain multiple-operation instructions, such as BLT and ILDB, to be interrupted before processing is completed.)

When an interrupt condition occurs, the monitor first determines if this type of condition is to cause a transfer of control to an interrupt servicing routine. If a transfer is to take place, the monitor transfers control to the location specified by your program's interrupt control block. If a transfer is not to take place, the condition's default action occurs. Figure 6-1 illustrates the software interrupt process.

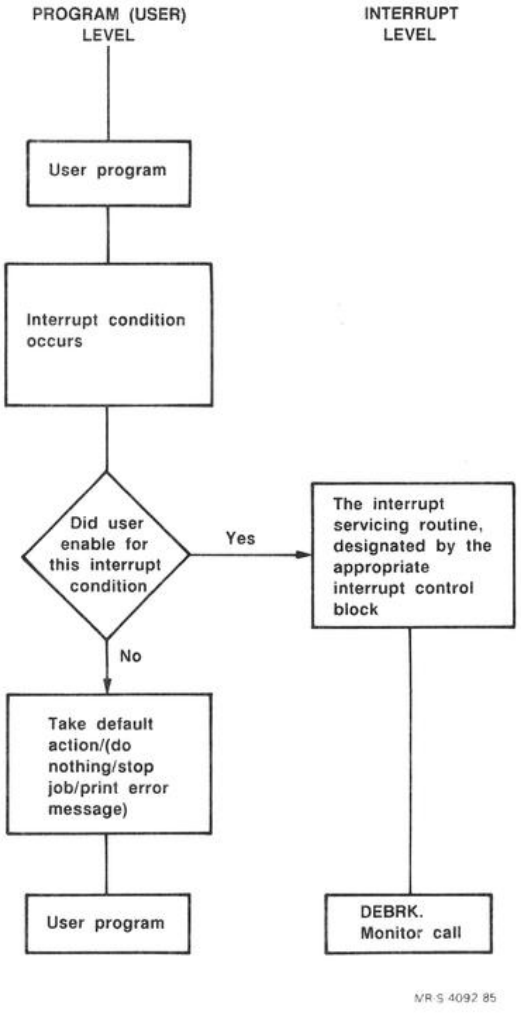


Figure 6-1: The Software Interrupt Process

When a transfer of control takes place, the monitor transfers control to your program's interrupt servicing routine. This action is called granting an interrupt request.

After an interrupt request has been granted, your program operates at interrupt level until it issues the DEBRK. monitor call. DEBRK. dismisses the interrupt servicing routine and causes any pending interrupt requests to be granted by the monitor. If there are no pending interrupt requests, your program is restarted at the point of interruption as though no interrupt had occurred. However, if the location containing the interrupted PC is changed during the interrupt handling, your program will return to a different location. If the interrupt occurred during the processing of a multiple-operation instruction, such as BLT, and the location containing the interrupted PC is changed, the remainder of the instruction is not completed.

When the monitor grants an interrupt request, the conditions that caused the interrupt are not changed. If the interrupt service routine that gains control after the interrupt does no further processing but only issues the DEBRK. monitor call, the result is the same as if the interrupt had not occurred. The monitor performs no special action on the condition (such as stopping a job on CTRL/C).

If an error interrupt occurs while the monitor is executing a monitor call for your program, the call is terminated. The only conditions that can cause interrupts during monitor call processing are error conditions in the calls. All other interrupt conditions (such as I/O completion) are deferred until the monitor call takes the error return or normal return.

To use software interrupts, your program must perform the following:

1. Initialize the software interrupt system. To do this, use the PIINI. UUO. PIINI. specifies the base address of an interrupt vector. The vector contains one or more 4-word interrupt control blocks, which control the PSI system. The PIINI. UUO clears any previous software interrupt system established by the program.

2. Turn on the PSI system with the PISYS. UUO. This call describes the conditions on which your program wishes control to be passed to an interrupt service routine and the offset to the appropriate interrupt control block within the interrupt vector specified in PIINI.

3. Contain an interrupt service routine to handle the specified interrupts. If control is to return to the main program, the interrupt service routine should end with the DEBRK. UUO. DEBRK. dismisses the software interrupt and returns control to the location where the interrupt occurred. Unlike APRENB and .JBINT trapping, you should not resume the interrupted program by using a JRSTF instruction.

6.3.1 PSI Monitor Calls

The following monitor calls are provided to allow you to write programmed interrupt service routines:

- PIINI. and PISYS.

  To initialize the PSI system for your program, you must specify the PIINI. UUO. You can specify extended (30-bit) addressing format by setting the PS.IEA bit in PIINI. To specify the conditions for interruption and the location of the interrupt handling routine, use PISYS. UUO.

- PIFLG.

  The PIFLG. UUO allows you to read or write PC flags at the time of the interrupt. Use PIFLG. when you have specified extended addressing in PIINI.. No flags are stored in the PC when extended addressing is in use.

- PIJBI.

  The PIJBI. UUO allows you to interrupt another job or JCH (Job Context Handle). The job you intend to interrupt must be enabled for the cross-interrupt (non-I/O) condition. If that job is currently handling an interrupt, the PIJBI. interrupt will fail, and your program should repeat its attempt to interrupt the job.

- PITMR.

  The program can be written to receive an interrupt after a specified amount of time. You must enable .PCTMR interrupts, and then use the PITMR. UUO to specify the amount of time after which to interrupt your job.

- PIBLK.

  Your program can examine the location of its interrupt control block when an interrupt is in progress by using the PIBLK. UUO. This is usually used by library modules that do not have direct control over the PSI vector and need to find out where it is.

- PISAV. and PIRST.

  Some programs may need to save and restore the PSI system's data base. For example, this may be used when calling another program using a GETSEG call, and the called program also wishes to use the PSI system.

  The PISAV. UUO returns the entire monitor data base for the PSI system to the program. This data base can be saved for reloading by using the PIRST. UUO, or can be analyzed to construct reports. The PISAV. UUO clears the current system after returning it to the program. Any interrupt conditions that may have occurred between the PISAV. and associated PIRST. call are lost

  The PIRST. monitor call restores (to the monitor) the data base for the PSI system; this data base is the one that was saved by a PISAV. call. The monitor checks the format of the restored block for consistency.

6.3.2 Interrupt Control Block

The interrupt control block is the controller of the PSI system. The control block keeps track of the following:

- The instruction that would have been executed next when the interrupt occurred.
- The location of the interrupt service routine to be used for processing the current interrupt.
- The reason for the current interrupt.

Your program can associate more than one interrupt condition with one interrupt control block. But the preferred usage is for your program to associate only one interrupt condition with an interrupt control block. An interrupt control block is represented in Figure 6-2.

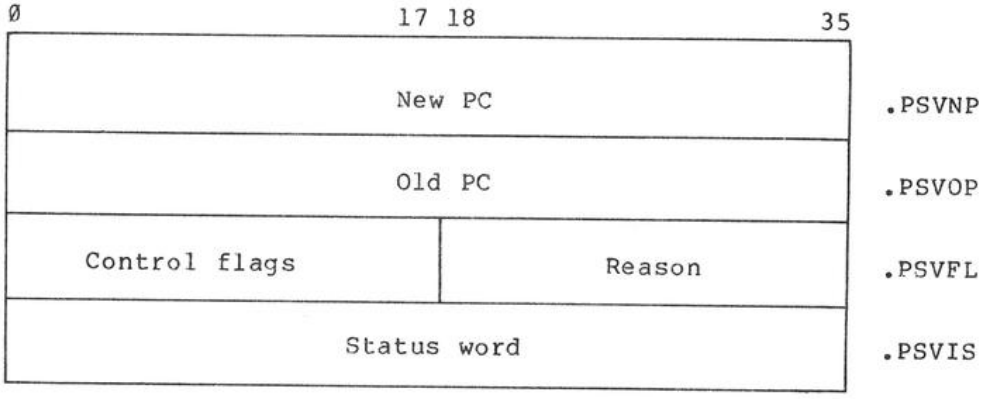


Figure 6-2: Interrupt Control Block

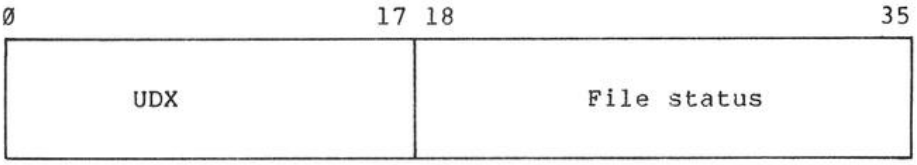
Where: **new PC** is the location of the interrupt servicing routine to be used for processing the current interrupt. This is a 30-bit PC if you set PS.IEA in the PIINI. UUO. Otherwise, the left half is ignored.

old PC is the current contents of the program counter. If you set PS.IEA in the PIINI. UUO, this is a 30-bit PC. You must use the PIFLG. UUO to read or write the flags. Otherwise, .PSVOP contains an 18-bit PC and flags. This value is equal to the address of the instruction after the location in your program where the interrupt occurred. If your program was in the process of executing a monitor call when the program received the interrupt, old PC contains the address of the call's return location (either error return or normal return). If, because there was an error condition in the call, the monitor call was terminated by the monitor, old PC contains the address of the monitor call, instead of its return address. This value is where your program will be resumed at the execution of a DEBRK. call. You can change the flow of the program by changing this value to another location within the program.

control flags are indicators specifying the circumstances under which an interrupt is to occur as well as how the monitor should treat the interrupt level code. This is what to do with other, possibly unrelated, interrupts while at interrupt level (refer to Table 6-2).

reason is the type of interrupt condition that has occurred; if BIT 18 is Ø, refer to Table 6-3, if Bit 18 is 1, refer to Table 6-4.

status word contains status information pertinent to the condition causing the interrupt. The information returned in the status word depends on the condition that caused the interrupt. For I/O conditions (see Table 6-3), the status word is returned as:



Where: **udx** is the Universal Device Index for the device specified in the interrupt condition. For disk devices, this is the channel number.

file status is the file status word (same as result of GETSTS monitor call).

If your program is enabled for interrupt conditions PS.RDO, PS.RIE, or PS.ROE on a network device, the monitor signals that the network node is no longer accessible by storing the "input error," "output error," and "device offline" status bits in the status word of the interrupt control block. Then the monitor generates the interrupt. This also occurs if I/O is attempted to a device on a CPU that has been DETACHed by the system operator.

Table 6-2: Control Flags

Bit	Symbol	Meaning
Ø		Reserved for use by DIGITAL.
1	PS.VPO	Disable all interrupts until your program re-enables them by using the PISYS. monitor call.
2	PS.VTO	Disable all interrupts of higher priority until your program issues a DEBRK. monitor call.
3	PS.VAI	Allow additional interrupts.
4	PS.VDS	Dismiss any additional interrupt requests for this condition (Table 6-4) or device that are received while an interrupt is in progress. This bit is useful if the interrupt service routine wants to perform functions that would cause another interrupt.
5	PS.VPM	Print the standard message, if one is relevant to this interrupt condition.
6		Reserved for use by DIGITAL.

6.3.3 Interrupt Conditions

The interrupt conditions are divided into two categories: I/O interrupts and non-I/O interrupts. You can specify an I/O condition for any device. The I/O conditions are listed in Table 6-3. The non-I/O conditions are listed in Table 6-4.

Table 6-3: I/O Interrupt Conditions

Bit	Symbol	Meaning	Bit	Symbol	Meaning
19	PS.RID	Input done	26	PS.RQE	Quota exceeded
2Ø	PS.ROD	Output done	27	PS.RWT	I/O wait
21	PS.REF	End-of-file	28	PS.ROL	Device on line
22	PS.RIE	Input error	29	PS.RRC	RIB has changed
23	PS.ROE	Output error	3Ø	PS.RDH	Device hung
24	PS.RDO	Device off line	31	PS.RSW	Reel switch
25	PS.RDF	Device full	32	PS.RIA	Input available

Table 6-4: Non-I/O Interrupt Conditions

Code	Symbol	Meaning
-1	.PCTLE	Your job's time limit has been exceeded. (Not applicable to batch jobs.) The runtime (in milliseconds) for your job is returned in the status word. You can issue the SET TIME command to alter your job's time limit.
-2	.PCTMR	A timer interrupt occurred.
-3	.PCSTP	You issued a CTRL/C. If your job is in terminal input wait state when this interrupt occurs, the monitor returns a 1 in Bit Ø of the status word. If the interrupt routine issues a DEBRK. call without altering the new PC, the program will not return to terminal input wait but will resume at the PC following the monitor call that caused the terminal input wait. This may not be desirable in some applications.
-4	.PCUUO	A monitor call is about to be processed. The status word contains the monitor call that was intercepted. If the interrupt routine issues a DEBRK. call without altering the old PC, the program will return to the intercepted monitor call and another interrupt will occur. Note that the only way to have the monitor call actually processed by the monitor is for the interrupt routine to execute the call itself. You cannot use .PCUUO to trap any of the PSISER UUOs (such as PIINI., PISYS., and DEBRK.).
-5	.PCIUU	An illegal monitor call has been executed. The status word contains the illegal monitor call.
-6	.PCIMR	An illegal memory location has been referenced. The status word contains the illegal effective address.
-7	.PCACK	An address check has occurred. The status word contains the device name.
-1Ø	.PCARI	An arithmetic exception has occurred.
-11	.PCPDL	A push-down list overflow has occurred.
-12	.PCNSP	One of the following occurred on DECnet: DECnet data became available, there was a logical link status change, or interrupt data became available. Refer to Chapter 5 for information about the NSP. UUO.
-13	.PCNXM	A non-existent memory location has been referenced.

Table 6-4: Non-I/O Interrupt Conditions (Cont.)

Code	Symbol	Meaning
-14	.PCAPC	A line frequency clock tick occurred. The status word contains the universal date/time word. This occurs only when your job is actually running; this does not occur every clock tick.
-15	.PCUEJ	A fatal error has occurred in your job.
-16	.PCXEJ	An external condition has caused a fatal error in your job.
-17	.PCKSY	A KSYS warning has occurred. The status word contains the minutes left until KSYS.
-20	.PCDSC	The dataset status has changed. The status word contains the new dataset status.
-21	.PCDAT	Either an ATTACH or DETACH monitor call has been executed. If DETACH, the status word contains a -1; if ATTACH, the status word contains the terminal's Universal Device Index.
-22	.PCWAK	A WAKE monitor call was executed. The status word contains the job number of the program that issued the wake. This interrupt is given only if the WAKE monitor call was actually issued while the job was hibernating.
-23	.PCABK	An address-break condition occurred.
-24	.PCIPC	Your job has received an IPCF packet in its input queue. The status word contains the associated variable: the length of the packet in the left half, and the flag word in the right half. The IPCF communications facility is described in Chapter 7.
-25	.PCDVT	A DECnet event occurred. The status word contains flags (DR.xxx) indicating the event.
-26	.PCQUE	An ENQ/DEQ resource is available for ownership. The status word contains the request-id of the request that was granted. If multiple requests were granted, the request-ids are ORed into the status word. Refer to Chapter 8 for more information.
-27	.PCNET	The ANF-10 network topology has changed. If your program receives this interrupt, it should issue a NODE. monitor call to determine the state of the network. This interrupt is useful for determining when a network node goes offline or comes online. This event occurs only in the ANF-10 network software.

Table 6-4: Non-I/O Interrupt Conditions (Cont.)

Code	Symbol	Meaning
-30	.PCJBI	A PIJBI. UUO was executed. The status word contains the job number or JCH in the left half and the value sent in the right half.
-31	.PCDTC	A date/time change occurred. The status word contains the universal date and time offset to be added.
-32	.PCOOB	An out-of-band character was received. The status word contains either the character and the udx of the TTY, or 400000,,udx.
-33	.PCRC1	Reserved for customer use.
-34	.PCRC2	Reserved for customer use.
-35	.PCSCS	An SCS. event occurred. Returned flags take the form SC%xxx. See the description of the .SSSTS word of the SCS. monitor call in Volume 2 for a list of the flags.
-36	.PCETH	An Ethernet event occurred. Flags returned are in the form ET.xxx. See the description of the .ETPSW word of the ETHNT. monitor call in Volume 2 for a list of the flags.

When an interrupt is granted to a program, the interrupt routine should investigate all possible causes for the interrupt. This is necessary because the amount of time that elapses between the event and the actual granting of the interrupt varies with system load and various scheduling parameters. For example, a single interrupt for network topology change (.PCNET) could represent several nodes appearing online or going offline.

It is possible for a condition enabled by a program to occur under circumstances where the program could not be granted an interrupt. In this case, the monitor acts as though the program were not enabled for these interrupts. This may cause the job to be stopped with an error message. The reasons an interrupt cannot be granted immediately are:

- The condition was caused by either the page fault handler or DDT.
- The interrupt system is not active (turned off).
- The condition occurred during an interrupt routine that is of equal or higher priority.

If the PSI vector is not useable, the monitor will halt your program and display the following message:

?PSI interrupt vector at addr for { CONDITION symbol / DEVICE name } is { write-protected / paged out / illegal }

In the message, addr refers to the address of the illegal vector. The condition symbol is the last three characters of the non-I/O condition as listed above (.PCxxx, where xxx is the symbol).

If you run your program in a non-zero section, you should set the PS.IEA bit in the PIINI. UUO to allow interrupting. If you have not, the monitor halts your program and displays the following message on an interrupt:

```
?Illegal non-zero section PSI interrupt at user PC addr
```

6.3.4 Example Using Programmed Interrupts

The following program shows how to use the PSI system to optimize disk I/O and compute-bound background activity:

```
TITLE PIEXMP - Example of programmed interrupts

;This program writes a file containing the integers
; from 0 to 100000 while doing a compute-bound background
; task.  Because the program never blocks for I/O,
; it can use all of the available CPU time.
; By using the programmed interrupt system, it drives
; the disk at full speed.

SEARCH  UUOSYM                          ;Symbol definitions

;ACs

T1==1                                   ;Work
N==2                                    ;Number to write on disk

;I/O channels

DSK=1                                   ;Disk file

;Initialization

START:  RESET
        MOVEI   T1,VECTOR               ;Get address of PI vector block
        PIINI.  T1,                     ;Initialize PI system
          HALT                          ;Not implemented
        OPEN    DSK,[UU.AIO+.IOBIN      ;Open disk for asynch binary output
                    SIXBIT /DSK/
                    XWD OB, 0]
          HALT                          ;Can't write
        MOVE    T1,[PS.FAC+[EXP DSK
                XWD 0,PS.ROD            ;Offset,,output done
                EXP 0]]                 ;Priority 0,,reserved
        PISYS.  T1,                     ;Enable for output done on disk
          HALT                          ;Failed
        MOVEI   N,0                     ;Initialize N
```

```
;Here on an output-done interrupt or at start of program

OUTDON: SOSGE   BYTECT              ;Room in this buffer?
        JRST    DMPBUF              ;No, go output buffer
        IDPB    N,BYTEPT            ;Yes, store in this buffer
        CAME    N,[^D100000]        ;Done?
        AOJA    N,OUTDON            ;No, back for next number
        CLOSE   DSK,                ;Yes, close the disk file
        EXIT                        ;Finished

;Here to output the buffer

DMPBUF: OUT     DSK,                ;Write out the buffer
          JRST  OUTDON              ;No errors and more buffers
        STATZ   DSK,IO.ERR          ;Any errors?
          HALT                      ;Fatal I/O error

;Here all available buffers are full, so we want to go back
; to the background task.

        DEBRK.                      ;Dismiss the interrupt
          HALT                      ;Can never get here

;Here if no interrupt was in progress.  We were called by
; initialization, and must now start the background task.

        MOVSI   T1,(PS.FON)         ;Turn on the PSI system so we can
        PISYS.  T1,                 ; get traps from background task
          HALT                      ;Can't turn it on
LOOP:   MOVEI   T1,0                ;Simple-minded background task
        AOJA    T1,LOOP             ;Do it again

;Buffer ring header

OB:     BLOCK   1
BYTEPT: BLOCK   1                   ;Byte pointer
BYTECT: BLOCK   1                   ;Byte count

;Interrupt vector

VECTOR: EXP     OUTDON              ;New PC
        EXP     0                   ;Old PC stored here
        EXP     0                   ;Flags
        EXP     0                   ;Status

END     START
```

CHAPTER 7

COMMUNICATING BETWEEN PROCESSES USING IPCF

The TOPS-10 interprocess communication facility (IPCF) allows jobs and programs on the same computer system to communicate with each other. This communication occurs when processes send and receive information in the form of packets. For the purposes of this description, a program is called a "process."

When a sender process sends a packet of information to a receiver process, the packet is placed in the receiver's input queue. The packet remains in the queue until the receiver checks the queue and retrieves the packet. Instead of periodically checking its packet input queue, the receiver can enable the PSI system to generate a software interrupt when a packet is placed in the input queue. Refer to Chapter 6 for details on the PSI system.

The following monitor calls allow you to use IPCF:

- The IPCFS. UUO sends an IPCF packet to another process.
- The IPCFR. UUO retrieves a packet sent by another process.
- The IPCFQ. UUO queries the IPCF input queue about your job.
- The IPCFM. UUO replaces a message exchange with a system process.

Any user without privileges can use the IPCF calls. A subset of functions are limited to privileged users, because IPCF services many communications needs of the monitor and the GALAXY batch and spooling system. Most functions enabling communication between user processes, however, do not require privileges. The IPCFM. monitor call is fully documented in Volume 2. The words and functions described in the following sections apply only to IPCFQ., IPCFR., and IPCFS..

7.1 PACKETS

Information is transferred in the form of packets from one process to another. Each packet is divided into two parts:

- A Packet Header Block (PHB) of four to six words in length.
- A Packet Message Block (PMB), which contains the actual message.

The PHB describes the characteristics of the communication (defines the sender and the receiver, for example) and points to the PMB, where the actual message is stored.

The PMB will be either a short-form block consisting of a few words (less than or equal to 12 (octal)) or a long-form block consisting of an entire memory page (1000 words). Use of the long-form block is also called "page mode." The default packet is a short block, but you can use the long form if you have privileges and if you set the appropriate flags bits in the packet header block (refer to Section 7.3).

For the short block, the monitor copies the data into an internal buffer to await a receiver. The short message length must be within the monitor's maximum, which is stored in the GETTAB table .GTIPC, item number 0 (%IPCML).

7.2 FORMAT OF THE PHB

The format of an IPCF Packet Header Block is as follows:

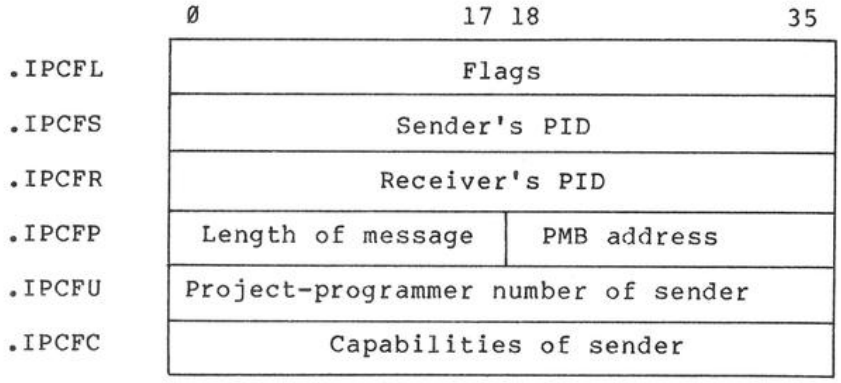


Figure 7-1: Packet Header Block

The PHB describes the sender of the message, the receiver of the message, and the location of the actual data message (Packet Message Block). It also contains flags that instruct the monitor to handle the communication of the data in different ways.

To set up the PHB, you may include the following words:

Word 0 (.IPCFL) contains instruction flags in the left half, and packet descriptor flags in the right half. Instruction flags are listed in Section 7.2.1. Descriptor flags are listed in Section 7.2.2.

Word 1 (.IPCFS) contains the sender's process identifier. For the sender, this word is filled by the monitor. For the receiver, one of the following may be specified in .IPCFS:

- The job number or JCH (Job Context Handle) of the sending process.
- The sending process's PID. (A PID is a unique Process IDentifier that you can obtain from [SYSTEM]INFO.)
- The address of the sender's PID. Setting the instruction flag IP.CFS in Word 0 allows you to indirectly reference the sender's PID. The monitor assumes that .IPCFS contains the location of the sender's PID.

- Zero. If you place a zero in this word, the monitor will assume one of the following:

  a. If your job has any PID(s), the monitor will choose one.

  b. If your job has no PID(s), the monitor will use the JCH for your current context.

In many cases, the job number sufficiently identifies a process, and a PID need never be used. The PID is used for programs that will be run from different jobs, so the program does not depend on job numbers. Refer to Section 7.8.2 for information about obtaining PIDs.

Word 2 (.IPCFR) contains the receiver's process identifier, which may be a job number, JCH, Ø, a PID, or address of PID (if IP.CFR is set in Word Ø). This word has the same characteristics as Word 1. For the receiving process, if this word is zero, the monitor fills it with the receiver's PID. Refer to Section 7.6 for more information about receiving packets.

Word 3 (.IPCFP) contains, in the left half, the length of the PMB, and, in the right half, the location of the first word in the PMB. For the short-form packet, the sender must include the actual length of the PMB, and the receiver must specify the maximum length of the PMB it is expecting, in the left half of this word. For the long-form packet, both sending and receiving PHBs must specify 1ØØØ in the left half of this word. For the long-form packet, this word must be page aligned.

Word 4 (.IPCFU) contains the PPN of the sending process. This optional word is ignored for sending packets. It is filled by the monitor on a receive, if reserved by the process.

Word 5 (.IPCFC) contains the sender's IPCF capability word. The IPCF capability word is described in Section 7.2.5.

7.2.1 IPCF Instruction Flags

The following instruction flags can be stored in the left half of Word Ø (.IPCFL) of the PHB. They are optional, and are listed here according to their bit positions.

Bits	Symbol	Meaning
Ø	IP.CFB	Do not block the receiver's job if there is no packet in the input queue. This bit is meaningful only when an IPCFR. monitor call is issued. If this bit is set when the IPCFR. is issued and there is no packet in the input queue, the IPCFR. call takes the error return and the monitor returns error code 3 (IPCNP%) in the AC. Use the HIBER monitor call or the PSI system (see Chapter 6) to notify the job when the packet arrives.
1	IP.CFS	Use the PID obtained from the address specified in .IPCFS as the sender's PID; this PID is called the indirect sender's PID.
2	IP.CFR	Use the PID obtained from the address specified in .IPCFR as the receiver's PID; this PID is called the indirect receiver's PID.

Bits	Symbol	Meaning
3	IP.CFO	Allow the sending process to send one packet more than the send packet quota. (This is called the "last phone call" bit.) This bit is meaningful only when an IPCFS. monitor call is issued. The default send quota is two. The quota is stored in GETTAB table .GTIPQ, bit field IP.CQS.
4	IP.CFT	Truncate the message if it is longer than the area reserved for it in .IPCFP. This flag is meaningful only when set for the IPCFR. monitor call. If the IPCFR. call does not set this bit and the packet in the input queue is larger than the area reserved for it, the IPCFR. monitor call takes the error return and the monitor returns error code IPCTL% in the AC. You can delete messages from your input queue by setting .IPCFP to Ø and IP.CFT to 1. This is a convenient way to delete long messages received when your program accepts only short messages.
5	IP.CFL	Allow a privileged program to send short-form packets that are larger than the installation's defined maximum. The system's absolute maximum is 51Ø (decimal) words. The preferred method for sending packets of this length is to use page mode (long-form packets).
6	IP.CRP	Receive packets only if they are addressed to the PID set in the .IPCFR word as specified in the IPCFR. UUO. (This is called a "PID-specific receive".) If this bit is not set, the monitor will simply return the next message received (in chronological order) for any PID owned by this job.

7.2.2 IPCF Packet Descriptor Flags

You can specify the following flags by setting the appropriate bits in the right half of .IPCFL, the first word in the packet header block.

Bits	Symbol	Meaning
7-17		Reserved for use by DIGITAL.
18	IP.CFP	This bit signifies that the program is privileged and intends to perform privileged functions. If this bit is not set, privileged functions will not succeed. If this bit is set, both processes must be privileged processes. If an unprivileged process sets this bit, the error return is taken from the monitor call and the monitor returns error code IPCPI% in the AC.
19	IP.CFV	The packet message is a page of data (1ØØØ words). Both the sender and the receiver must set this bit. (Refer to Section 7.3 for more information about using long-form messages.)

Bits	Symbol	Meaning
20	IP.CFZ	Send a packet with a packet header block but no packet message block; this type of packet is called a zero-length packet.
21	IP.CFA	The sender of the packet requires the receiver to acknowledge receipt of the packet. The monitor does not ensure that the acknowledgement is made.
22-23		Reserved for use by DIGITAL.
24-29	IP.CFE	The field for an error code sent by a privileged process ([SYSTEM]INFO and [SYSTEM]IPCC). These error codes (70 through 77) are listed in Volume 2, in the description of the IPCFR. UUO. An error code in this field indicates that the monitor call executed properly and the normal return was taken with no errors returned in the AC. However, the sending process (for example, [SYSTEM]INFO) is returning an error, such as "duplicate name." If this field is empty, no error occurred.
30-32	IP.CFC	The field for the system sender code. This code can be set by a privileged process only; however, the monitor will return the code so that an unprivileged process can examine it. The system sender codes are:

Code	Symbol	Meaning
1	.IPCCC	The packet was sent by [SYSTEM]IPCC.
2	.IPCCF	The packet was sent by system-wide [SYSTEM]INFO.
3	.IPCCP	The packet was sent by the receiver's local [SYSTEM]INFO.
4	.IPCCG	The packet was sent by [SYSTEM]GOPHER, a privileged process that sends [SYSTEM]IPCF message types 40 and 50 listed in Section 7.8.3. It does not accept packets from user programs, but often conducts dialogs with certain system programs. GOPHER is an active task; IPCC is a passive task.

Bits	Symbol	Meaning
33-35	IP.CFM	The packet in the input queue is a packet that was returned to the sender. If IP.CFM is equal to 1 (.IPCFN) the packet was returned to the sender because the PID was destroyed before the packet was received but after the packet was sent. IP.CFM can only be set by a privileged process. The monitor returns the packet so that a nonprivileged process can examine it.

7.2.3 Process Identifiers

Words 1 (.IPCFS) and 2 (.IPCFR) of the PHB are reserved for identifying the sender and the receiver of the packet. When sending a packet, .IPCFS can be zero. When receiving a packet, .IPCFR can be zero.

Job numbers may be used to identify a process. However, if your job has more than one process, or your job is communicating with more than one process, the PID uniquely identifies each process. PIDs are assigned by [SYSTEM]INFO, and may or may not be accompanied by a symbol name.

PIDs are assigned to a process by [SYSTEM]INFO. There can be any number of PIDs assigned to a process, up to the maximum PID quota, which by default is two. Each PID is unique and never reused until the system is reloaded; in this case, all previous PID assignments are cleared. Both communicating processes should agree on the PIDs and symbolic names being used. This facility releases the programs from system-specific characteristics that may change from execution to execution, such as job numbers.

The method for sending packets to [SYSTEM]INFO is described in Section 7.8.2.

7.2.4 Symbolic Names

Symbolic names are specified by the process. A process specifies the name in the PMB when it requests a PID from [SYSTEM]INFO. [SYSTEM]INFO assigns a PID to the process and associates the symbolic name with the PID. [SYSTEM]INFO will not allow the assignment of a name that is already assigned to another PID, unless the owner of that name makes the request.

A symbolic name must be an ASCIZ string up to 29 characters long. Therefore, the maximum size of the name is six data words terminated by a zero byte. The symbolic names to be used should be agreed upon in advance by the communicating processes.

A symbolic name can be written in one of the following formats:

Format	Example
text	ASCIZ /CORPORATION/
[project,programmer]text	ASCIZ /[27,2407]TEST/
text[project,programmer]	ASCIZ /TEST[26,7077]/
text['ANY',programmer]	ASCIZ /EXAMP['ANY',5332]/
['ANY',programmer]text	ASCIZ /['ANY',2433]FOO/
text['ANY','ANY']	ASCIZ /ACCT['ANY','ANY']/
['ANY','ANY']text	ASCIZ /['ANY','ANY']WONDER/
text[project,'ANY']	ASCIZ /FILE[10,'ANY']/
[project,'ANY']text	ASCIZ /[23,'ANY']CLASS/
[SYSTEM]text	ASCIZ /[SYSTEM]GOPHER/
text[SYSTEM]	ASCIZ /GOTO[SYSTEM]/

The wildcard character * can be substituted for the phrase 'ANY'. Note that the following strings are different:

Format	Example
Name[PPN]	ASCIZ/FOO[10,10]/
[PPN]Name	ASCIZ/[10,10]FOO/

When a PPN is used as part of the symbolic name, it must be the PPN under which the process is currently running. [SYSTEM] can only be specified as part of the symbolic name of a privileged process.

If a process wants to send a message to another process but it knows only the process's name and not its PID, the sending process can ask [SYSTEM]INFO for the PID associated with the name. Note, however, that the list of PIDs and symbolic names that [SYSTEM]INFO keeps is cleared when [SYSTEM]INFO or the system is reloaded.

7.2.5 IPCF Capability Word

Word 5 (.IPCFC) of the PHB contains flags representing the privileges of the sender. Ignored when specified for a sending process, this word will be filled by the monitor on a receive, if this word was reserved. The sender capability bits are:

Bits	Symbol	Meaning
0	IP.JAC	The sender of the packet is running with the JACCT bit set; it is a privileged process.
1	IP.JLG	The sender of the packet is logged-in.
2	IP.SXO	The sender of the packet is an execute-only job.
3	IP.POK	The sender of the packet has POKE privileges.

Bits	Symbol	Meaning
4	IP.IPC	The sender of the packet has privileges allowing him to perform special, privileged IPCF functions. Note that this requires your program to have JP.IPC set in the job privilege word.
5-17		Reserved for DIGITAL.
18-26	IP.SCN	Sender's context number
27-35	IP.SJN	This field contains the job number of the sender.

7.3 LONG-FORM MESSAGES

A packet can take either of two forms: short (normal) or long. A short message has a PMB of Ø to 12 (octal) words. A long message has a PMB consisting of one page (1ØØØ octal words) and can be sent using page mode. Only one page can be sent per packet. To use page mode, both sending and receiving processes must:

- Set the left half of Word 3 (.IPCFP) of the PHB to be 1ØØØ.
- Specify the page number in the right half of .IPCFP.
- Set Bit 19 (instruction flag IP.CFV) in Word Ø (.IPCFL) of the PHB.
- During an IPCFS., the page is removed from your program's core image. You can create a new page in its place using the PAGE. UUO.
- During an IPCFR., a page is created at the virtual address specified in the right half of .IPCFP. It is assumed that a page has been reserved there for this purpose. If the page already exists, the IPCFR. UUO fails. Your program can ensure that the page is available by destroying it with the .PAGCD function of the PAGE. UUO (with PA.GAF set), before using IPCFR. If the page number is in a section that does not exist, the section map will be created automatically.

7.4 QUOTAS

For each process, outstanding IPCF messages are counted for sending and receiving. The number of packets that have been received are counted and this count is stored in GETTAB Table 1Ø5 (.GTIPP) in word IP.CQO. The number of packets that have been sent but have not been received is stored in the same table in IP.CQP. Both the send queue and the receive queue are limited to a maximum number of packets that can be outstanding: these are the send quota and the receive quota. This quota cannot be exceeded, and when the maximum is reached, the process must wait until a packet has been sent/retrieved from its queue.

The default quotas allow two messages outstanding in the send queue and five messages outstanding in the receive queue. However, the system manager at each installation can set IPCF quotas on a per-user basis. If these quotas are zero, the process cannot use IPCF.

The send packet quota and the receive packet quota are stored in GETTAB Table 77 (.GTIPQ). The send quota is stored in field IP.CQS and the receive quota is stored in field IP.CQR.

The quotas are used when the job sends packets using the IPCFS. UUO. Error code 1Ø (IPCFRS%) is returned in the AC when the sender queue is full. The program should attempt to discover why the packets have not been sent, and, resolving that, try to send the current packet again. Error code 11 (IPCFRR%) is returned in the AC when the IPCFS. UUO fails because the receiver's queue is full. The program should handle this error by building a resend queue, so that it can keep trying to send the packet.

7.5 SENDING AN IPCF PACKET USING IPCFS. UUO

Any process can send a packet to another process using the IPCFS. UUO. The calling sequence for IPCFS. is show below:

```
            MOVE    ac,[XWD len,addr]   ;Point to PHB
            IPCFS.  ac,                 ;Send the packet
              error return              ;Something wrong
            normal return               ;Continue
            .   .   .
addr:       flags                       ;PHB
            sender's PID
            receiver's PID
            XWD len2,addr2
            .   .   .
addr2:      message                     ;PMB
```

In this calling sequence, **len** is the length of the PHB, and **addr** is the address of the PHB. The PHB includes your own PID and the the receiver's PID. Also in the PHB, len2 and addr2 are the length and address of the PMB, and message is the actual packet of data. The PMB is a short-form message.

7.6 RETRIEVING AN IPCF PACKET USING IPCFR. UUO

For a process to retrieve a packet from its input queue, the process must issue the IPCFR. UUO. To retrieve a packet, the process does not have to know who sent the packet. The IPCFR. UUO merely checks the IPCF receive queue for any waiting packets and retrieves the packet that has been in the queue the longest period of time.

The calling sequence for the IPCFR. UUO is shown below:

```
            MOVE    ac, [XWD len,addr]  ;Set up call
            IPCFR.  ac,                 ;Retrieve the packet
              error return              ;Something wrong
            normal return               ;Continue
            .   .
addr:       flags                       ;PHB
            Ø
            Ø
            XWD len2,addr2
            .   .   .
addr2:      BLOCK 12                    ;PMB
```

In this calling sequence, len is the length of the PHB and addr is the address for storing the PHB of the retrieved message. In the PHB, neither sender's nor receiver's PID need to be specified. Also in the PHB, len2 and addr2 point to where the actual message will be stored.

Although the sender's and receiver's PID (.IPCFS and .IPCFR in the PHB) need not be specified for the IPCFR. UUO, there are cases in which it is very useful to include these. Using PID-specific receives, it is possible to retrieve a message from a specific process, rather than the packet that is next in the queue.

On a normal return from the IPCFR. UUO, the **associated variable** is returned in the AC. The associated variable describes the next packet in the process's input queue, if there is one. It contains the length of the next packet in the queue in its left half, and the right half of the flag word (descriptor flags) of the PHB of the next packet in its right half. The associated variable is also stored in the PSI status word when a software interrupt is generated (refer to Chapter 6).

The associated variable is used to check the receive queue for the next message, and the type of message that is waiting.

7.7 QUERYING THE NEXT IPCF PACKET USING IPCFQ. UUO

You can query the IPCF receive queue for your job using the IPCFQ. UUO. This call returns the PHB for the next packet in your queue, but includes the number of packets in your queue instead of the address of the next packet's message block.

The value of the IP.CFV bit in the returned PHB tells your job whether to expect a long or short message block on the next IPCFR. call. The more efficient programming technique, however, is to do an IPCFR. with the expectation of receiving either a long or short packet. If the IPCFR. call fails, your program can toggle the IP.CFV bit and repeat the IPCFR. UUO. This reduces the number of monitor calls that your program makes, substantially improving performance. Most programs need never use the IPCFQ. UUO.

To query the next IPCF packet in your input queue, use the IPCFQ. monitor call as shown:

```
        MOVE    ac, [len,addr]          ;Set up call
        IPCFQ.  ac,                     ;Query the next packet
          error return                  ;Something wrong
        normal return                   ;Continue
        .   .   .
addr:   flags                           ;PHB
        Ø
        Ø
        XWD len2,n
```

In this calling sequence, len is the length of the argument list and addr is the address for storing the retrieved packet. The PHB contains zero for sender and receiver PIDs. Word 3, .IPCFP, contains len2, the length of the next packet, and n, the number of messages in the receive queue.

Note that IPCFQ. does not return the next packet in the queue; it returns only information about it. Your program can identify the sending process on an IPCFQ. call as it would for IPCFR., by including the sender's PID in Word 1 (.IPCFS) and setting the flag IP.CRP in the flag word.

If there is no packet in the queue, IPCFQ. takes the error return, with error code 3 (IPCNP%) in the AC.

7.8 SYSTEM PROCESSES

There are two system processes of general interest: [SYSTEM]INFO and [SYSTEM]IPCC. The [SYSTEM]INFO process is the information center for the Inter-Process Communication Facility. This process performs system functions related to process identifiers, and any IPCF process can request these functions by sending packets to [SYSTEM]INFO. [SYSTEM]INFO is described in Section 7.8.1.

[SYSTEM]IPCC is the IPCF controller, and it performs many packet controlling functions. Privileged processes can request IPCC functions by sending packets to [SYSTEM]IPCC. Unprivileged processes are limited in the functions they can request from [SYSTEM]IPCC. [SYSTEM]IPCC is described in Section 7.8.2.

7.8.1 [SYSTEM]INFO

[SYSTEM]INFO is the information center for the Inter-Process Communication Facility. Any IPCF process can request [SYSTEM]INFO to perform a function for it. A process requests a function by sending [SYSTEM]INFO a packet, and [SYSTEM]INFO responds to the request by sending a packet back to the initiating process. The initiating process obtains the response to its requested function by issuing the IPCFR. call to retrieve the packet sent to it by [SYSTEM]INFO. If the process plans to block for the [SYSTEM]INFO response, the IPCFM. UUO may provide a more convenient interface. (Refer to the IPCFM. monitor call description in Volume 2.)

The calling sequence of sending a request to [SYSTEM]INFO is shown below:

```
                    MOVE    ac,[XWD len,addr]
                    IPCFS.  ac,
                      error return
                    normal return
                    .   .   .
addr:               flags
                    Ø
                    Ø
                    XWD len2,addr2+.IPCIØ
                    .   .   .
addr2+.IPCIØ:       XWD ack-code,fcn-code
addr2+.IPCIl:       Ø or duplicate PID
addr2+.IPCI2:       argument
```

In the calling sequence, the PHB is set up as described in Section 7.2. Note that the PID for [SYSTEM]INFO is always Ø.

The PMB specifies the information that [SYSTEM]INFO requires, as follows:

ack-code is a unique code you may supply. This code is returned unchanged in the response from [SYSTEM]INFO. If a process has sent more than one request to [SYSTEM]INFO, the user code provides a method of determining which response is for which request.

function-code is a [SYSTEM]INFO function code. The [SYSTEM]INFO function codes are listed below.

duplicate PID is the PID of the process that you would like to receive a duplicate copy of the response from [SYSTEM]INFO. If no process is to receive a duplicate copy, this value should be zero.

argument is the argument to the function code. The argument depends on the function code, and these are listed below.

The calling sequence to retrieve a packet sent by [SYSTEM]INFO is shown below:

```
            MOVE      ac,[len,,addr]
            IPCFR.    ac,
              error return
            normal return
            .   .   .
addr:       flags
            Ø
            Ø
            XWD len2,addr2
            .   .   .
addr2:      BLOCK 12
```

To receive a packet from [SYSTEM]INFO, set up the PHB in the same way as you constructed it for the IPCFS. UUO. Location **addr** specifies the beginning of the packet header block, and **addr2** specifies the beginning of the packet message block. The response from [SYSTEM]INFO depends on the function code you specified in the PHB. In general, the PMB returned by [SYSTEM]INFO takes the following form:

Word	Contents
addr2+.IPCIØ/	ack-code,,fcn-code
addr2+.IPCI1/	PID
addr2+.IPCI2/	symbolic name

Because this information depends on the function you specified, the PMB returned for each function is described with the function codes listed below. Words not needed for [SYSTEM]INFO's response will remain unchanged. The functions recognized by [SYSTEM]INFO are:

Fcn-code	Symbol	Meaning
1	.IPCIW	Requests [SYSTEM]INFO to return the PID associated with the specified symbolic name. The format of the request is: addr2: XWD user-code,.IPCIW PID for copy or zero symbolic name The format of the response is: addr2: XWD user-code,.IPCIW PID for name symbolic name
2	.IPCIG	Requests [SYSTEM]INFO to return the symbolic name associated with the specified PID. The format of the request is: addr2: XWD user-code,.IPCIG PID for copy or zero PID The format of the response is: addr2: XWD user-code,.IPCIG PID in request name for PID
3	.IPCII	Requests [SYSTEM]INFO to assign a PID to the calling process and to associate the symbolic name with the newly assigned PID. The format of the request is: addr2: XWD user-code,.IPCII PID for copy or zero symbolic name The format of the response is: addr2: XWD user-code,.IPCII PID symbolic name Any PID obtained from [SYSTEM]INFO as a result of an .IPCII request is disassociated from the calling job when the job performs a RESET. PIDs obtained with this function code have the format: 4nnnnn,,nnnnnn. To obtain a PID without a symbolic name, simply zero the word at addr2+2.

Fcn-code	Symbol	Meaning
4	.IPCIJ	Requests [SYSTEM]INFO to assign a job-wide PID. This differs from .IPCII in that the PID requested with .IPCIJ will not be cleared on a RESET. It is retained until you destroy the context or log out. The format of the request is: addr2: XWD user-code,.IPCIJ PID for copy or zero symbolic name The format of the response is: addr2: XWD user-code,.IPCIJ PID symbolic name Any PID obtained from [SYSTEM]INFO as a result of an .IPCIJ request is disassociated from the calling job only when the job logs off the system. PIDs obtained with this function code have the format: Ønnnnn,,nnnnnn. A job can have more than one PID and symbolic name assigned to it. However, there is a maximum number of PIDs that can be assigned to a job. If a request is made for a PID and the PID quota has been filled, the flag word of the response from [SYSTEM]INFO contains error code 73 in the error code field. The PID quota is stored in GETTAB table .GTIPC, Item %IPCDQ.
5	.IPCID	Requests [SYSTEM]INFO to disassociate the specified PID from its job number. This function can be requested only by the owner of the specified PID or by an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCID PID for copy or zero PID to be dropped The format of the response is: addr2: XWD user-code,.IPCID Ø PID that was dropped

Fcn-code	Symbol	Meaning
6	.IPCIR	Requests [SYSTEM]INFO to disassociate all PIDs that were created by .IPCII and are associated with the specified job number. This function can be requested only by the owner of the job or JCH, or an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCIR PID for copy or zero job-number or JCH The format of the response is: addr2: XWD user-code,.IPCIR Ø job-number or JCH
7	.IPCIL	Requests [SYSTEM]INFO to disassociate all PIDs that were created by .IPCIJ and are associated with the specified job number. This function can be requested only by the owner of the job or JCH, or an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCIL PID for copy or zero job-number or JCH The format of the response is: addr2: XWD user-code,.IPCIL Ø job-number or JCH
1Ø	.IPCIN	Requests notification from [SYSTEM]INFO when a specified PID has been disassociated from a job. The format of the request is: addr2: XWD user-code,.IPCIN PID for copy or Ø PID The format of the response is: addr2: XWD user-code,.IPCIN Ø PID
15	.IPCIS	Used only by [SYSTEM]IPCC on the execution of the LOGOUT and RESET monitor calls.

7.8.2 [SYSTEM]IPCC

[SYSTEM]IPCC is the IPCF controller. Only privileged processes can request all [SYSTEM]IPCC functions. A privileged process requests a function by sending [SYSTEM]IPCC a packet, and [SYSTEM]IPCC responds by sending a packet back to the initiating process. The initiating process obtains the response to its requested function by issuing the IPCFR. UUO to retrieve the packet sent to it by [SYSTEM]IPCC. The IPCFM. UUO provides a more convenient interface to [SYSTEM]IPCC. Since [SYSTEM]IPCC functions always complete immediately, using IPCFM. saves one UUO execution per request to [SYSTEM]IPCC.

To perform IPCF privileged functions and to send packets to [SYSTEM]IPCC, a process must have the JACCT bit set, be running under [1,2], or have the IPCF privilege bit set. The IPCF privilege bit is in GETTAB table .GTPRV, Bit Ø, JP.IPC. IP.CFP must also be set in the PHB.

The format of sending a request to [SYSTEM]IPCC is:

```
                        MOVE      ac,[XWD len,addr]
                        IPCFS.    ac,
                          error return
                        normal return
                        .   .   .
addr:                   flags
                        sender's PID
                        receiver's PID
                        XWD len2,addr2+.IPCSØ
                        .   .   .
addr2+.IPCSØ:           XWD ack-code,function-code
addr2+.IPCS1:           argument1
addr2+.IPCS2:           argument2
addr2+.IPCS3:           argument3
```

In an IPCC request, the PHB is set up as described in Section 7.2. [SYSTEM]IPCC's PID, to be stored in Word 2 (.IPCFR), can be obtained from GETTAB table 126, Item Ø, %SIIPC.

The PMB to [SYSTEM]IPCC depends on the function code being used. In general, Word Ø contains the **ack-code**, a code you supply to determine which response is for which request, and the **function-code**, one of the [SYSTEM]IPCC function codes, which are listed below.

The type of argument and number of argument words is different depending on the function code used. Therefore, they are described with the function codes, listed below.

The format for retrieving a packet sent by [SYSTEM]IPCC is:

```
            MOVE      ac,[len,,addr]
            IPCFR.    ac,
              error return
            normal return
            .   .   .
addr:       flags
            Ø
            Ø
            len2,,addr2
            .   .   .
addr2:      BLOCK 12
```

Where the PHB is similar to that described in Section 7.2. When retrieving a packet, it is not necessary to specify the receiver's and sender's PID.

The response, starting at addr2, will be in the following general format:

Word	Contents
addr2+.IPCSØ/	ack-code,,fcn-code
addr2+.IPCS1/	[SYSTEM]IPCC response
addr2+.IPCS2/	.
addr2+.IPCS3/	.

In the response, the ack-code and fcn-code are those you specified in your PMB, and are returned unchanged for your verification. The rest of [SYSTEM]IPCC's response depends on the function you specified. Responses for each function code are listed below. Words not needed for [SYSTEM]IPCC's response will remain unchanged.

[SYSTEM]IPCC Function Codes

Fcn-code	Symbol	Meaning
-n		Negative function codes are reserved for use by customer programs.
1	.IPCSE	Requests [SYSTEM]IPCC to enable the specified job number to receive IPCF packets. This function can be requested only by an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCSE job-number The format of the response from [SYSTEM]IPCC is identical to the format of the request.
2	.IPCSD	Requests [SYSTEM]IPCC to disable the specified job from being able to receive IPCF packets. This function can be requested only by an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCSD job-number The format of the response from [SYSTEM]IPCC is identical to the format of the request.

Fcn-code	Symbol	Meaning
3	.IPCSI	Requests [SYSTEM]IPCC to return the PID associated with [SYSTEM]INFO. Unprivileged processes may request this function. The PID returned is for the local [SYSTEM]INFO (if there is one) or the global [SYSTEM]INFO, if there is no local [SYSTEM]INFO. This PID may also be obtained from GETTAB table .GTSID, item %SIINF. The format of the request is: addr2: XWD user-code,.IPCSI The format of the response from [SYSTEM]IPCC is: addr2: XWD user-code,.IPCSI PID for [SYSTEM]INFO
4	.IPCSF	Requests [SYSTEM]IPCC to create a PID for [SYSTEM]INFO. This function can be requested only by an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCSF m n In the PMB, m is the PID of a process to make [SYSTEM]INFO and n is the job number for which to create a local (job-specific) [SYSTEM]INFO. To create a PID for a global [SYSTEM]INFO, n should be Ø. If n is a job number, a PID for a local [SYSTEM]INFO is created for the specified job. Local [SYSTEM]INFOs are valid only if another (local or global) does not exist. If m is Ø, the specified [SYSTEM]INFO is deleted. A local [SYSTEM]INFO can be created only by another local [SYSTEM]INFO or by a global [SYSTEM]INFO. If there is no local [SYSTEM]INFO, any privileged job can create one. The global [SYSTEM]INFO can be changed or destroyed only if the calling process is [SYSTEM]INFO. The format of the response is identical to the format of the request.
5	.IPCSZ	Requests [SYSTEM]IPCC to clear a specified PID. This function may be requested by an unprivileged process for your own PID. The format of the request is: addr2: XWD user-code,.IPCSZ PID to be destroyed The format of the response is identical to the format of the request.

Fcn-code	Symbol	Meaning
6	.IPCSC	Requests [SYSTEM]IPCC to create a PID for a specified job number or JCH. This function is unprivileged for your job or JCH, and within your PID quota. The format of the request is: addr2: XWD user-code,.IPCSC type,,job-number (or JCH) The format of the response is: addr2: XWD user-code,.IPCSC type,,job-number (or JCH) PID In the PMB, you specify type by setting/clearing Bit Ø. When Bit Ø is set, the PID created by this function is valid until the job performs a RESET. When Bit Ø is clear, the PID created by this function is valid until the job logs off the system. In the left half of addr2+1, you specify the job-number for which the PID is desired. [SYSTEM]IPCC returns the PID for the specified job in addr2+1.
7	.IPCSQ	Requests [SYSTEM]IPCC to set a send and a receive quota for the specified job. This function can be requested only by an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCSQ PID or job-number On a response to this function, [SYSTEM]IPCC returns the send quota in Bits 18-26 of addr2+.IPCS2 and the receive quota in Bits 27-35 of addr2+.IPCS2.
1Ø	.IPCSO	Requests [SYSTEM]IPCC to change the job number associated with the specified PID. This function can be requested only by an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCSO PID new-job-number (or JCH) The format of the response is identical to the format of the request.

Fcn-code	Symbol	Meaning
11	.IPCSJ	Requests [SYSTEM]IPCC to return the JCH associated with the specified PID. Unprivileged processes may request this function. The format of the request is: addr2: XWD user-code,.IPCSJ PID The format of the response is: addr2: XWD user-code,.IPCSJ PID JCH
12	.IPCSP	Requests [SYSTEM]IPCC to return the PIDs associated with the specified job number or JCH. Unprivileged processes may request this function. The number of PIDs returned depends on the length of the reserved argument block. The format of the request is: addr2: XWD user-code,,.IPCSP addr2+1: job-number or JCH addr2+2: starting PID or Ø The PIDs are returned starting with addr2+2 in the form: addr2: user-code,.IPCSP job number or JCH PID . . . addr+n: PID
13	.IPCSR	Requests [SYSTEM]IPCC to return the send and receive quotas associated with the specified job number or specified PID. Unprivileged processes may request this function. The format of the request is: addr2: XWD user-code,.IPCSR job-number or PID The send quota is returned in Bits 18-26 of addr2+2 and the receive quota is returned in Bits 27-35.
14	.IPCSW	Obsolete
15	.IPCSS	Reserved for DIGITAL.

Fcn-code	Symbol	Meaning
16	.IPCQS	Sets the PID quota of the target specified in the request to the value given at addr2+2. A target can be a job number, a job-context handle (JCH), or a PID. This function can be requested only by an IPCF privileged process. The format of the request is: addr2: XWD user-code,.IPCQS target to be set PID quota The response takes the same form as the request.
17	.IPCQR	Requests [SYSTEM]IPCC to read the PID quota of the specified target. Unprivileged processes may request this function. The format of the request is: addr2: XWD user-code,.IPCQR target The response takes the form: addr2: XWD user-code,.IPCQR target PID quota
20-22		Reserved to DIGITAL.
23	.IPCLP	Requests [SYSTEM]IPCC to locate a given PID in the special system PID table (listed below in .IPCRP). Unprivileged processes may request this function. The format of the request is: addr2: XWD user-code,.IPCLP PID to locate The response takes the form: addr2: XWD user-code,.IPCLP PID Index of the located PID
24	.IPCWP	[SYSTEM]IPCC will write the table of special system PIDs (listed below). This function can be requested only by an IPCF privileged process. The request is in the form: code,,.IPCWP offset PID

Fcn-code	Symbol	Meaning
25	.IPCRP	[SYSTEM]IPCC will read the special system PID table. Unprivileged processes may request this function. The request is in the form: code,,.IPCRP offset The response takes the form: code,,.IPCRP offset PID Special system PIDs are:

Code	Symbol	Meaning
Ø	.IPCPS	[SYSTEM]IPCC
1	.IPCPI	[SYSTEM]INFO
2	.IPCPQ	[SYSTEM]QUASAR
3	.IPCPM	Mountable Device Allocator
4	.IPCPT	Tape Label Process
5	.IPCPF	File Daemon
6	.IPCPC	Tape Automatic Volume Recognition Process
7	.IPCPA	[SYSTEM]Accounting
1Ø	.IPCPO	Operator Interface
11	.IPCPL	System Error Logger
12	.IPCPD	Disk Automatic Volume Recognition Process
13	.IPCPE	[SYSTEM]TGHA
14	.IPCNM	Network Management (NCP)
15	.IPCPG	[SYSTEM]GOPHER
16	.IPCPV	[SYSTEM]CATALOG
17	.IPCPX	[SYSTEM]MAILER

NOTE

The following message types are sent by [SYSTEM]GOPHER or [SYSTEM]IPCC to and from GALAXY components.

26	.IPCSU	[SYSTEM]IPCC will send a message to [SYSTEM]QUASAR indicating that a spooled file is closed.
27	.IPCSL	[SYSTEM]IPCC will send a logout message to [SYSTEM]QUASAR. The job-number is returned in addr2+1.
3Ø	.IPCTL	[SYSTEM]IPCC sends a tape labeling message.
31	.IPCUO	A mountable unit is on-line. (IPCC)
32	.IPCON	LOGIN message sent to [SYSTEM]QUASAR. (IPCC)
33	.IPCAC	Accounting messages. (IPCC)
34	.IPCDE	MDA-controlled device deassigned. (IPCC)
35	.IPCME	MDA memory error. (IPCC)

Fcn-code	Symbol	Meaning
36	.IPCCS	Reserved.
37	.IPCRS	Reset with locked structure to MDA. (IPCC)
40	.IPCQU	QUEUE UUO to MDA. (GOPHER)
41	.IPCLC	Search list change to MDA. (IPCC)
42	.IPCAT	Primary disk port attach to MDA. (IPCC)
43	.IPCDT	Primary disk port detach to MDA. (IPCC)
44	.IPCXC	Disk unit exchange to MDA. (IPCC)
45	.IPCRM	Structure removal to MDA. (IPCC)
46	.IPCMT	Magtape unit accessible to MDA. (IPCC)
47	.IPCST	Structure mount to MDA. (IPCC)
50	.IPCIM	IPCFM. UUO request to [SYSTEM]INFO. (GOPHER)

The following is an example program using IPCF communication to obtain information about the GALAXY input and output queues.

```
                        TITLE IPCF demonstration

;This program demonstrates how to acquire a named PID
; from [SYSTEM]INFO, and, using that PID, request a queue
; listing from QUASAR.  The QUASAR communications logic
; is similar to, but much less complex than that used
; in the QUEUE CUSP.

        SEARCH  UUOSYM                  ;For TOPS-10 UUO symbols
        SEARCH  GLXMAC,QSRMAC           ;For GALAXY and QUASAR symbols

        T4=1+<T3=1+<T2=1+<T1=1>>>       ;Define some ACs to use
        P=17                            ;PHL pointer

        PDLSIZ==30                      ;Push down list length
        PHBLEN==6                       ;Packet header block length
        PMBLEN==12                      ;Packet message block length
        NAMLEN==<D29/5>+1               ;Max length of a name in words
        ACKCOD==171004                  ;Initial ack code
        PAGSIZ==1000                    ;Length of a page in words
        PAG==400                        ;Page number for page receives

IPCF:   JFCL                            ;No CCL entry
        RESET                           ;Stop the world
        MOVE    P,[IOWD PDLSIZ,PDL]     ;Set up push down list pointer
        MOVEI   T1,ACKCOD               ;Get initial ack code
        MOVEM   T1,ACK                  ;Save it for later use
        MOVE    T1,[%SIQSR]             ;Argument to return a PID
        GETTAB  T1,                     ;Get QUASAR's PID
          SETZ  T1,                     ;Assume QUASAR isn't running
        JUMPE   T1,NOQSR                ;Is it?
        MOVEM   T1,QSR                  ;Save the PID
        PUSHJ   P,GENNAM                ;Generate a name
        PUSHJ   P,GETPID                ;Get a PID from [SYSTEM]INFO
        PUSHJ   P,SNDQSR                ;Send queue list request to QUASAR
        PUSHJ   P,RCVQSR                ;Receive and list the queues
        EXIT                            ;Return to monitor level
```

```
;Generate an ASCIZ name to associate with our PID.
; To insure the name is unique, our job number will
; be appended to the end of the name text.

GENNAM: SETZM   NAM                     ;Clear out
        MOVE    T1,[NAM,,NAM+1]         ;Name text
        BLT     T1,NAM+NAMLEN-1         ;Storage
        MOVE    T4,[POINT 7,NAM]        ;Set up byte ptr to storage
        MOVE    T2,[POINT 7,TXT]        ;Set up byte ptr to initial text
GENNA1: ILDB    T1,T2                   ;Get a character
        JUMPE   T1,GENNA2               ;End of text?
        IDPB    T1,T4                   ;Put a character
        JRST    GENNA1                  ;Loop through text string
GENNA2: PJOB    T1,                     ;Get our job number
        SETZ    T3,                     ;Clear a counter
GENNA3: IDIVI   T1,12                   ;Divide by 10 (decimal)
        PUSH    P,T2                    ;Save remainder
        SKIPE   T1                      ;Done?
        AOJA    T3,GENNA3               ;No--recurse
GENNA4: POP     P,T1                    ;Get a digit
        ADDI    T1,"0"                  ;Make it ASCII
        IDPB    T1,T4                   ;Append to name
        SOJGE   T3,GENNA4               ;Loop for all digits
        POPJ    P,                      ;And return

;Get a named PID from [SYSTEM]INFO.  This routine uses
; the name text generated by GENNAM.

GETPID: SETZM   PHB+.IPCFL              ;Clear first word of PHB
        SETZM   PHB+.IPCFS              ;Default our PID
        SETZM   PHB+.IPCFR              ;0 is [SYSTEM]INFO's PID
        MOVE    T1,[PMBLEN,,PMB]        ;Load length,,addr of PMB
        MOVEM   T1,PHB+.IPCFP           ;Point to PMB
        SETZM   PHB+.IPCFU              ;No PPN
        SETZM   PHB+.IPCFC              ;Zero capability word
        AOS     T1,ACK                  ;Add one to make unique ack-code
        HRLZS   T1                      ;Set up ack-code
        HRRI    T1,.IPCII               ;Request PID with name
        MOVEM   T1,PMB+.IPCI0           ;Load ack,,fcn-code into PMB
        SETZM   PMB+.IPCI1              ;Zero second word of PMB
        MOVE    T1,[NAM,,PMB+.IPCI2]    ;Pointer to load name into PMB
        BLT     T1,PMB+.IPCI2+NAMLEN-1  ;Load name into PMB
        PUSHJ   P,IPCSND                ;Send packet
GETPI1: PUSHJ   P,IPCRCV                ;Retrieve packet
        HLRZ    T1,PMB+.IPCS0           ;Get ack-code of packet
        CAME    T1,ACK                  ;Compare with ack of sent packet
        JRST    GETPI1                  ;Acks not equivalent, try again
        MOVE    T1,PMB+.IPCS1           ;Get our PID
        MOVEM   T1,PID                  ;Store our PID
        POPJ    P,                      ;Return
```

```
;Build and send a message to QUASAR.  A "normal" queue
; listing will be requested  (that is, the same listing obtained
; by issuing the QUEUE monitor command).

SNDQSR: SETZM   PHB+.IPCFL          ;No IPCF flags
        MOVE    T1,PID              ;Get our PID
        MOVEM   T1,PHB+.IPCFS       ;Make it the sender's PID
        MOVE    T1,QSR              ;Get QUASAR's PID
        MOVEM   T1,PHB+.IPCFR       ;Make it the receiver's PID
        MOVE    T1,[PMBLEN,,PMB]    ;Get length and address of PMB
        MOVEM   T1,PHB+.IPCFP       ;Point the monitor at the PMB
        MOVE    T1,[PMBLEN,,.QOLIS] ;Length,,msg type (queue listing)
        MOVEM   T1,PMB+.MSTYP       ;Save it
        SETZM   PMB+.MSFLG          ;Send no flags to QUASAR
        AOS     T1,ACK              ;Get a new ack code
        MOVEM   T1,PMB+.MSCOD       ;Save for comparison later
        MOVEI   T1,1                ;One data block in the message
        MOVEM   T1,PMB+.OARGC       ;Save count in message
        SETZM   PMB+.OFLAG          ;Request a normal queue listing
        MOVE    T1,[2,,.LSQUE]      ;Queue block
        MOVEM   T1,PMB+.OHDRS+ARG.HD  ;Save queue block header
        SETOM   PMB+.OHDRS+ARG.DA   ;Save queue block data
        PUSHJ   P,IPCSND            ;Send queue listing request
        POPJ    P,                  ;Return

;Receive a queue listing from QUASAR.  This routine contains
; no provisions for handling multiple queue  listing messages
; which can occur when there are many jobs in the queues.

RCVQSR: MOVEI   T1,IP.CFV           ;Returned message is a page
        MOVEM   T1,PHB+.IPCFL       ;Save flag
        MOVE    T1,[PAGSIZ,,PAG]    ;Length and page where to put msg
        MOVEM   T1,PHB+.IPCFP       ;Point monitor at it
RCVQS1: PUSHJ   P,IPCRCV            ;Try to receive a msg
        MOVE    T1,<PAG_^D9>+.MSCOD ;Get ack code
        CAME    T1,ACK              ;Make the one we sent out?
        JRST    RCVQS1              ;No--try again
        MOVEI   T1,<PAG_^D9>+.OHDRS ;Point to start of data in msg
RCVQS2: HRRZ    T2,ARG.HD(T1)       ;Get a block type
        CAIN    T2,.CMTXT           ;Is this the queue listing text?
        JRST    RCVQS3              ;Yes--go output it
        HLRZ    T2,(T1)             ;No--get this block length
        ADDI    T1,(T2)             ;Offset to the next block
        JRST    RCVQS2              ;Keep searching
RCVQS3: OUTSTR  ARG.DA(T1)          ;Output listing text
        POPJ    P,                  ;Return
IPCSND: MOVE    T1,[PHBLEN,,PHB]    ;Load length,,addr of PHB
        IPCFS.  T1,                 ;Send packet
          SKIPA                     ;Always skip on failure
        POPJ    P,                  ;Return
        OUTSTR  [ASCIZ |? Error sending packet|]  ;Print error
        EXIT                        ;Bomb out

IPCRCV: MOVE    T1,[PHBLEN,PHB]     ;Load length,,addr of PHB
        IPCFR.  T1,                 ;Get packet
          SKIPA                     ;Always skip on failure
        POPJ    P,                  ;Return
        OUTSTR  [ASCIZ |? Error receiving packet|]  ;Print error
        EXIT                        ;Bomb out
```

```
NOQSR:  OUTSTR  [ASCIZ  |? Cannot get QUASAR's PID|]  ;Print error
        EXIT                            ;Bomb out

TXT:    ASCIZ   |IPCF demo Job  |       ;Symbolic name

PDL:    BLOCK   PDLSIZ                  ;Push down list
PHB:    BLOCK   PHBLEN                  ;Packet header block
PMB:    BLOCK   PMBLEN                  ;Packet message block
NAM:    BLOCK   NAMLEN                  ;Name to assign to PID
ACK:    BLOCK   1                       ;Ack code
PID:    BLOCK   1                       ;Our PID
QSR:    BLOCK   1                       ;QUASAR's PID

        END     IPCF
```

CHAPTER 8

RESOURCE CONTROLS: THE ENQ/DEQ FACILITY

When several users access the same file, problems of interference and inconsistency can arise. While one user is reading the file, other users can also read that file; but no other user should be writing the same portion of the file. And while a user is writing the file, no other user should be reading or writing the same portion of the file.

For example, suppose a group of users have agreed that a character string of the form:

```
CUSTMR.DATnnnn
```

represents a block in the file CUSTMR.DAT, where **nnnn** is the block number. Then if one user has obtained exclusive use of block 14 in that file, perhaps so that he can write the block, the monitor will not grant other requests for use of the same block until the user releases it.

The ENQ/DEQ facility can be used for dynamic resource allocation, computer networks, and internal monitor queueing. Simultaneous file access, however, is its most common application. The ENQ/DEQ facility ensures data integrity among jobs, allowing multiple users to share resources, and it ensures synchronism among cooperating jobs.

ENQ/DEQ ensures data integrity among jobs only when the participating jobs cooperate when using both the facility and the resource. The facility does not prevent non-cooperating jobs from accessing a resource without first enqueueing it. However, to enqueue a resource, the requesting user must have access to the resource. The accessibility of the resource depends on the type of resource. If, for example, the resource is a file, access to the file is permitted or denied depending on the access protection code of the file (see Section 12.3).

A resource is an entity within the system. Jobs compete with one another to use the resource. The physical resource itself has no relationship with the resource definition supplied to the ENQ/DEQ facility by the requesting programs. Competing jobs, however, are synchronized to allow controlled access to resources by the ENQ/DEQ facility. The ENQ/DEQ facility uses the resource definitions supplied by cooperating programs to arbitrate use of a resource. Some examples of resources are files, operations on files (such as reading and writing), records, devices, and memory pages.

RESOURCE CONTROLS: THE ENQ/DEQ FACILITY

The ENQ/DEQ facility maintains a queue of requesting jobs for each resource that has been enqueued (requested) by any job. You request resource ownership by placing a request in the queue associated with that resource. You make this request for a resource using the ENQ. monitor call. An ownership request indicates that you want the ENQ/DEQ facility to create a lock between your job and the resource you have defined. Each request in the queue must be satisfied before following requests can be considered. When you obtain a lock with a resource, your are the "owner" of the resource until you dissolve the lock using the DEQ. monitor call to dequeue the resource.

In the ENQ. call to request a resource, you specify information about the resource itself, the type of ownership you require, and information about the way the request is to be handled. Each ENQ. request enters the queue associated with the resource. The queue is an ordered list of all requests for that resource.

When the monitor grants a lock to a requesting job, it establishes the type of lock that the job specified. For each resource, the first owner of the resource determines the characteristics of the lock and the resource. While the lock is in effect, the requesting job is the owner of the resource and can use the resource. All other jobs requesting access to the resource must specify the same resource identifier. The resource identifier is an ASCIZ string or numeric value that is included in the ENQ. call.

If the owner of the resource has defined the lock to be sharable, other jobs can be granted a sharable lock on the resource without waiting for the first owner to relinquish the resource. Without an explicit definition, the first owner's lock is assumed to be exclusive, and other jobs must wait in the queue for the previous owner to relinquish the resource. When the first owner relinquishes the resource, the next requesting job is granted a lock. If the ENQ. call by the requesting job specifies a sharable lock, the lock will be granted to subsequent jobs that also request shared ownership. Therefore, to share the ownership of a resource, all sharing owners must cooperate in the shared ownership agreement. Section 8.1.1 discusses sharable resources.

You relinquish ownership of the resource by using the DEQ. call. This call can also be used to remove a waiting request from the resource queue.

This cycle of enqueueing and dequeueing requests for resource ownership continues until all requests have been granted for that resource. After the last job relinquishes the resource, the monitor deletes the resource queue and all data associated with the resource. When a new job makes a request for the resource, a new queue is created for the resource.

8.1 REQUESTING A RESOURCE

The ENQ. call places a request in the resource queue for the resource that your program defines in the ENQ. argument list. This definition is an arbitrary ASCIZ text string pointed to by a word in the ENQ. argument list, or a numeric value included in the ENQ. argument list. This resource definition governs the queue into which the request is placed. Therefore, cooperating programs must use the same resource definition when competing for the same resource. This resource definition is an arbitrary value to the monitor, and is not used to actually prevent or allow access to any physical resource.

As each request for the resource is made, the request is placed at the end of an ordered list of requests for the resource. Depending on the types of requests in the queue, a lock may or may not be granted to the requesting job immediately. The first owner of the resource can specify sharable access to the resource. This allows subsequent requests for sharable access to the same resource to be granted immediately. If the first owner does not specify sharable access, the lock is assumed to be exclusive, and no further lock on the resource can be granted until the owner relinquishes the resource.

8.1.1 Sharable Resources

Sharable access is useful when multiple jobs must access a resource (such as a file) in a non-modifying mode (such as reading the file). When reading files, multiple jobs can share ownership of the resource without interfering with one another's data. As long as the sharing programs cooperate in ensuring data integrity, they can share ownership without endangering one another. If a request is made for exclusive ownership, that request and all subsequent requests must wait until all the sharing jobs have relinquished the resource. When the last sharing owner of the resource has dequeued the resource, the owner requesting exclusive ownership (who may intend to write to the file) is granted a lock on the resource. Any jobs making requests after the exclusive request must wait until the exclusive lock is dissolved by the owner. Therefore, your ENQ. requests should specify sharable access unless you must modify the resource.

The ENQ/DEQ facility allows you to limit the set of users that can share a resource at the same time. You provide a sharer group number in your ENQ. argument list. When your job is the owner of the resource, only other jobs specifying the same sharer group number can access the resource. Again, note that jobs in the resource queue are satisfied in order. Thus, a request for sharable access with the same sharer number must follow the first owner's request without any intervening requests of another type.

Sharer group Ø is the default sharer group. Thus, every request that specifies sharable access, without specifying a group number, is a member of sharer group Ø. To restrict access to a sharable resource, a sharer group number other than Ø must be specified.

8.1.1.1 Resource Pools - A resource pool is a group of identical resources (such as magtape drives) or copies of the resource (such as memory pages). You specify the resource pool in the argument list to the ENQ. call by specifying the number of units or copies of the resource that are in the pool. You also specify the number of units or copies of the resource to which your job requires exclusive access.

A pooled resource cannot be requested for sharable access. That is, a resource may be either sharable or pooled, but never both. When the owner has exclusively locked a certain number of units from the resource pool, subsequent jobs can gain exclusive access to the rest of the units or copies in the pool. Each subsequent job must specify the total number of units in the pool and the number of units to which it requires access. As long as each request specifies the same pool size, the resource is pooled, and units are subtracted from the pool according to the number of units requested by each job in the queue. The ENQ/DEQ facility ensures that requests for pooled resources specify the same pool size (that is, the same number of units or copies in the pool) and that requests are granted for the number of units or copies available (unowned) in the pool.

Pooled resources can be actual physical units, such as magnetic tape drives. A certain number of these might be available on the system; this number is the pool size. Each job requesting access to tape drives must request a number of drives equal to or less than the total pool size. A pooled resource might be only one actual unit (such as a disk file) to which a limited number of users can be allowed access at one time. The disk file can be specified as a pool by a requestor of the resource, and subsequent access to the file is limited to the number of copies of the file specified as the pool size, minus the number of copies requested for ownership from the pool.

When the number of resources in the pool has been determined, the ENQ/DEQ facility allocates the resources until the pool is depleted, or until a request is made for more units in the pool than are available. In the latter case, the job making the request is not granted ownership of any resources until enough resources have been dequeued by other jobs to satisfy the request. Because requests are satisfied in the order they are queued, all subsequent requests must wait for the previous request to be satisfied. As jobs relinquish resources, the resources are returned to the pool of available resources. When all resources have been returned, the monitor deletes the resource pool. The next request for a pooled resource redefines the pool size and available number of units or copies in the pool.

A pooled resource is useful when a limited number of jobs wish to modify the same resource at the same time. If the resource were a file, the jobs would be simultaneously updating the file. Of course, if there is no limit to the number of jobs modifying the resource, there is no need to use the ENQ/DEQ facility.

8.1.1.2 Partitioned Resources - A resource can be exclusively accessed by more than one job if those jobs require a portion of the resource. The resource is requested in partitions, thus allowing several users access to the resource, but with the intention of modifying restricted and exclusive sections of the resource. This is specified using a bit mask that defines the partition.

For example, a user might require access to block 14 of a file CUSTMR.DAT, but only for certain records in that block. Another user might require access to a different record in the same block of the same file. Each request can specify a bit pattern, where bits that are set correspond to parts of the resource to be locked and bits that are off denote portions that will be available to other jobs at the same time.

If a job requests parts of a resource that are independent of parts of the resource already owned by another job, a request for exclusive access to the resource can be granted. Therefore, the bit mask specified in the ENQ. request should specify unique bit masks for each portion of the resource. In the case of a disk file, each bit in the bit mask might correspond to a record. Thus, if a request is made for exclusive access to a portion of a resource already owned by another job, the bit masks would overlap. The requests would be conflicting, and the second request would wait until the owner had relinquished the resource. If the bit masks did not overlap, the records would be mutually exclusive, and the second request could be granted at the same time that the resource is owned by the first job.

| Since the monitor transfers one block at a time for I/O, simultaneous attempts to write to the same physical block of a file may cause data corruption. Refer to Section 8.3 for information on passing data to other jobs.

8.1.2 Multiple-Lock Requests

If your job requires access to several resources at one time, you may place several lock requests in a single ENQ. call. All of the requests must be granted to your job before the ENQ. call is successfully returned. The lock requests in the ENQ. call must be granted in the order that you specify the resources in the call.

When your job issues a multiple-lock request, the first request is considered before the subsequent requests. Your job is placed in the queue for the first resource until the first lock is granted. Then your job is placed in the queue for the second resource, and so forth. The requests that have not yet been considered are "invisible" to the monitor. Other jobs requesting the same resource as the invisible requests in your call can be granted access ahead of your job, until the request for that resource becomes visible (that is, until all previous requests made by your ENQ. call are granted).

8.1.2.1 ENQ. Quotas - A multiple-lock request must not request more resources than your job's ENQ. quota allows. The ENQ. quota is the total number of requests that your job can have at one time. Your job cannot use the ENQ. facility if its ENQ. quota is Ø.

Your system administrator sets ENQ. quotas. If you need a larger quota, see your system administrator. You can check the quota for your job by using the .ENQCG function of the ENQC. monitor call.

You can obtain the default ENQ. quota from item %EQDEQ in GETTAB table .GTENQ.

8.1.2.2 Request Levels - Each request in a multiple lock request is granted in the order that it is presented in the ENQ. call. Each request in the call must be assigned a level number, unless you bypass level checking, by setting the flag EQ.FBL. In this case, it is recommended that you use deadlock detection. The level number for the first request in the call must be equal to or greater than the level number of any previous request for the same resource. Subsequent requests in the call must have ascending level numbers.

The level numbers you include in each request for a resource in the same ENQ. call help the ENQ/DEQ facility to avoid deadlocks between jobs. A deadlock occurs when two or more jobs are each waiting for resources held by other jobs, and no job can relinquish its own resources until it is granted access to those resources owned by the other jobs. You can avoid deadlocks if you always request the same resources in the same order. This is accomplished using level numbers. Your job must use a level number for each resource it is requesting, and all jobs must use the same level numbers for the same resources. The jobs must request the resources in ascending order, and relinquish the resources in descending order. This ensures that your job will not be granted a lock for a resource until all lower-level requests have been granted first. Resources are best utilized if the scarcest or most-requested resources are requested with higher level numbers.

8.1.3 Granting Locks

Requests for resources are granted on a first-come first-served basis. Multiple-request calls must be granted in the order that the resources are requested, and the call is returned only when all the resources have been locked for the job. By default, jobs must wait for the requests to be granted.

However, there are methods for avoiding the wait. The ENQ/DEQ facility allows you to use the PSI system, a time limit, or a deadlock detection flag to prevent undue interruption of processing.

8.1.3.1 ENQ. Software Interruption - You can enable the PSI system to interrupt your program when a request is granted for your job. To use the PSI system, you should first consult Chapter 6.

Non-blocking jobs should use the ENQ. function .ENQSI to enqueue a request and to continue execution. If all the requests in the call can be granted at once, the call is returned successfully. If any or all requests cannot be granted, the ENQ. call takes the error return, returning the error code ENQRU% in the accumulator. The instruction in your program at the non-skip return from the ENQ. call should branch to a routine that can be processed while your program waits for the request(s) to be granted.

When the monitor interrupts your job, your program should check the status word of the PSI interrupt control block. The interrupt reason for ENQ. requests granted is .PCQUE.

To use the PSI system for ENQ. requests, your program should include a request identifier for each request in the ENQ. argument block. This request-id is associated with the resource being requested, and is returned when that resource is granted, in the status word of the interrupt control block. When you make a multiple-request ENQ. call, using the PSI system to interrupt your program when the requests are granted, the request-ids of the granted requests are inclusively ORed into the status word. Therefore, for such a request, it is advisable to use bit masks for the request-ids, specifying a single bit for each request in the call. This allows you to check which requests have been granted.

8.1.3.2 Time Limits - Your program can avoid waiting an undue length of time without enabling the PSI system. If you have a specific time limit within which the request must be granted, you can include this time limit (specified in seconds) in the header block of the ENQ. argument list. If the requests in your call can be granted immediately, the call returns successfully. If not, the job waits the specified number of seconds for each request that cannot be immediately granted. When that time is over, and all the requests in the call have not been granted, the call takes the error return, returning error code ENQTL% in the accumulator.

8.1.3.3 Deadlock Detection - If your program makes multiple requests with multiple calls, you can avoid creating a deadlock situation by including the deadlock detection flag in your ENQ. call. If you set this flag, your requests are compared against other requests for the same resource, and the possibility of a deadlock is determined. In the case that the requests can be granted without causing a deadlock, the call returns successfully. However, if granting your requests would cause a deadlock, the call takes the error return, and the code ENQDD% is returned in the accumulator.

If you specify both a time limit and deadlock detection, the monitor first waits until the request times out before checking for deadlocks. This avoids the overhead of deadlock detection for the majority of cases where the job is being blocked, but the resource will be freed before the time limit is up.

You can use this feature to implement a form of deadlock priority. For example, a batch job could specify a long wait time, and an interactive job could specify a short wait time. The interactive job, which wants quick response, will time out first in the case of a deadlock, and will back out of its transaction. The batch job, which probably has more time invested in its transaction, can afford to wait longer. It continues processing when the interactive job releases its resources.

8.2 RELEASING RESOURCES

Resources are released when you relinquish them using the DEQ. monitor call. The resources are also released if your program issues a RESET, EXIT, or LOGOUT call. If your job issues a CLOSE on the channel for which ENQ. locks are in effect, the call fails, setting the I/O status bit IO.IMP.

When resources are relinquished, they are freed according to level number, in descending order. That is, resources locked first are released last. If your job is the only owner of any resource, the queue and data for the resource are eliminated when you relinquish the resource, to be reset by any new request for the resource. However, your program can ensure that resource queues and data are preserved by using a long-term lock or an eternal lock.

Normally the ENQ/DEQ facility retains its data for defined resources only while one or more jobs have locks or requests for those resources; when the last request is dequeued, the monitor deletes the data.

However, the overhead for deleting and redefining resource data can be eliminated by using a long-term lock. A lock is "long-term" if the EQ.FLT flag is set in the ENQ. call argument list for the resource. When the last locking job releases the resource, the monitor retains the resource data for approximately five minutes, instead of deleting the data immediately. Thus, when another job requests the resource, the resource data is still in the ENQ. data base.

You can also define the lock as an "eternal lock." An eternal lock prevents the resource from being automatically dequeued when your program issues a RESET call. The resource will only be relinquished when you explicitly relinquish it with DEQ.

8.3 PASSING DATA TO OTHER JOBS

You can pass data to other jobs sharing ownership of the same resource owned by your job. The data is in the form of a lock-associated block; its content is arbitrary to the monitor, and is meaningful only to the participating programs.

A lock-associated block is defined in the ENQ. argument list. The block definition and data persist until either of the following occurs:

- Another job redefines it.
- The monitor deletes its data for the resource. This occurs immediately when there are no further requests for the resource (or five minutes later, for a long-term lock).

A lock-associated block may be lost too soon if its queue does not have a long-term lock. Therefore, it is good programming practice to use long-term locks when using lock-associated blocks.

The lock-associated block can also be used for local buffer caching (also called distributed buffer management). Local buffer caching allows a number of jobs to maintain copies of data (for example, disk blocks), in buffers local to each job. The job can be notified when the buffers contain invalid data due to modifications by another job. In applications where modification is infrequent, substantial I/O may be saved by maintaining local copies of buffers (hence the names "local buffer caching" or "distributed buffer management").

To support local buffer caching using the lock-associated block, each job maintains a cache of buffers with no locks on resources that represent the current contents of each buffer. If the buffer contains diskblocks, the lock-associated block associated with each resource are used to contain a disk block version number. The first time a lock is obtained on a particular disk block, the current version number of that disk block is returned in the job's lock-associated block. If the contents of the buffer are cached, this version number is saved along with the buffer. To re-use the contents of the buffer, the resource associated with the buffer must have a long-term lock put on it, in either shared or exclusive mode, depending on whether the buffer will be read or written. The lock-associated block returned with the lock contains the latest version number of the disk block. The version number of the disk block is compared with the saved version number. If they are equal, the cached copy is valid. If they are not equal, a fresh copy of the disk block must be read from disk.

Whenever a procedure modifies a buffer, it writes the modified buffer to disk and then increments the version number in the lock-associated block prior to dequeueing the lock on the resource associated with the buffer. This way, the next job that attempts to use its local copy of the same buffer will find a version number mismatch and must read the latest copy from disk, rather than use its cached and now invalid buffer.

If more than five minutes have passed since the last user of the resource dequeued the lock, then no lock-associated block data will be returned, and the program should invalidate its buffer anyway.

8.4 ENQ/DEQ MONITOR CALLS

The ENQ. facility offers three monitor calls:

- The ENQ. call requests access to a resource. When the requested resource is not immediately available, the request is queued until the resource is released. When the resource is available, the request is granted, and a lock is placed on the resource. The request defines the characteristics of the lock (for example, whether the resource is sharable; that is, may be accessed simultaneously by another process).

- The DEQ. call releases a resource, cancelling the lock on the resource, or cancels requests for a resource.

- The ENQC. call allows you to obtain the request quota for a job, change the request quota, and dump information about the monitor's data base of ENQ/DEQ resources. Note that some of these operations require privileges.

8.5 BUILDING REQUESTS

When your program uses an ENQ., DEQ., or ENQC. call, it must have already constructed the argument block for the call, which defines the resources. This consists of a header block for the entire list of requests and one lock block for each resource. Together, the header and all associated lock blocks are known as the request block.

The format of the header block is:

Word	Symbol	Contents
Ø	.ENQLL	Number of blocks and length:

Bits	Symbol	Meaning
Ø-5	EQ.BHS	Header block size
6-17	EQ.LNL	Number of lock blocks.
18-35	EQ.LLB	Total length of the request block.

Word	Symbol	Contents
1	.ENQRI	Request identifier.
2	.ENQTL	Time limit (number of seconds).

The header block size (EQ.BHS) gives the length of the header block (1 to 3 words). The default size (if you give the size as Ø) is 2.

The number of lock blocks (EQ.LNL) is the number of lock blocks in the list that follows.

The length of the argument list (EQ.LLB) gives the total length of the request block. Note that all the lock blocks in a single ENQ. request must be the same length. Thus, the value of EQ.LLB must be the header block size plus the length of each lock block times the number of lock blocks.

The request identifier (.ENQRI) is an optional identifier for the request. The left half of .ENQRI must be Ø. The request-id is stored in the right half of this word. If you use the ENQ/DEQ facility with programmed interrupts, an interrupt caused by the availability of a resource returns the inclusive OR of the request-ids of resources that have become available, in the status word for the PSI system. (Refer to Section 8.6.3.)

The time limit (.ENQTL) is an optional word in the header block that you can use to specify a maximum amount of time for the job to block while waiting for a .ENQBL function to be granted. You specify the number of seconds for your job to wait for requests to be granted before returning from the ENQ. call. (This word is used only by the ENQ. UUO.) When the time limit is exceeded, the monitor returns the error code ENQTL%.

One or more lock blocks follow the header block. Each lock block requests a lock for one resource. All the lock blocks in a single request block must be the same length.

The format for each lock block is:

Word	Symbol	Contents
Ø	.ENQFL	Flags, level, and channel:

Bits	Symbol	Meaning
Ø	EQ.FSR	Sharable lock.
1	EQ.FBL	Don't do level checking.
2	EQ.FLT	Long-term lock.
3	EQ.FEL	Eternal lock, released only on request.
4	EQ.FAB	Aborted lock. This flag is useful on a modification function (.ENQMA) to prevent the resource from being locked by any other jobs.
5	EQ.FDD	Enable deadlock detection.
6	EQ.FCW	Specifies that .ENQBP contains a 36-bit user code.
7-8		Reserved.
9-17	EQ.FLV	Level number.
18-35	EQ.FCC	Channel number (if positive integer) or code (if negative integer). The codes are:

Code	Symbol	Meaning
-3	.EQFPL	Privileged global (system-wide) lock. This lock code can be used only by jobs logged in under [1,2] or jobs with the JACCT privilege. Thus, only other jobs with the [1,2] or JACCT privileges can share the lock.

Word	Symbol	Contents			
			Code	Symbol	Meaning
			-2	.EQFGL	Global lock. This flag prevents any sharers, regardless of privileges, and requires that you have JP.ENQ privileges set in your job's privilege word (GETTAB table .GTPRV).
			-1	.EQFJB	Job-wide lock. This flag prevents other requests by your job (including other contexts) for this resource from being granted.

1	.ENQBP	Resource identifier in the form of a numeric value or a pointer to an ASCIZ string.
		
		Bits Ø to 2 (EQ.BUC) have the value 5 if a 33-bit numeric value is used. The word contains a 36-bit numeric value if EQ.FCW is set in the flag word (.ENQFL). The number is any arbitrary string that is specified by all cooperating processes.
		
		Otherwise, .ENQBP contains a pointer to an ASCIZ string of up to 3Ø words (15Ø characters) that is the name of the resource. .ENQBP cannot use indirection or indexing to address the name string. This pointer is either a byte pointer of the form:
		
		`POINT 7,address,bitplace`
		
		where bitplace is the location of the rightmost bit in the first byte in the first word; and address is the address of the first word;
		
		or a pointer of the form:
		
		`XWD -1,address`
		
		where address is the address of the first word of the string. For the second form, the string must be 7-bit bytes and begin in the first byte of the first word.			

2	.ENQPS	Pool size or sharer group. If a resource pool is being specified, the left half of this word (EQ.PPS) contains the total pool size, and the right half (EQ.PPR) contains the number of units or copies desired from the pool. If a sharer group is specified, the left half (EQ.PPS) contains Ø and the right half (EQ.PPR) contains the sharer group number. This optional word defaults to zero.			

Word	Symbol	Contents
3	.ENQMS	Mask for partitioned lock, where the left half (EQ.MBL) is the mask length, in words, and the right half (EQ.MSK) is the address of the mask. This optional word defaults to zero.
4	.ENQTB	Lock-associated block, where the left half (EQ.TLN) contains the length of the block, in words. The right half (EQ.TBL) contains the address of the block. This optional word defaults to zero.

The ENQ. header/lock block structure looks as follows:

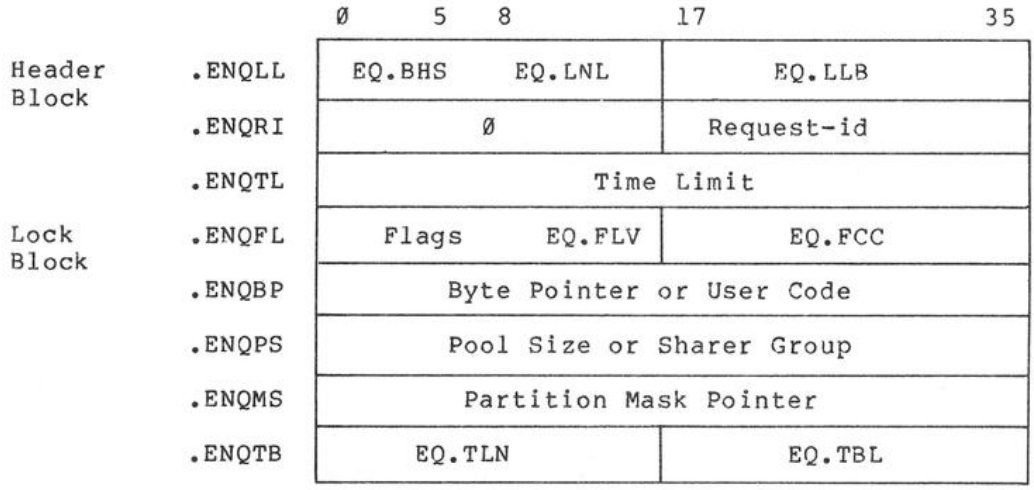


Figure 8-1: ENQ/DEQ Request Block

Note that the lock block is relative to the end of the header block, which may vary in size, because the request-id (.ENQRI) and the time limit (.ENQTL) are optional. The lock block is repeated for each resource in the request.

8.6 QUEUEING REQUESTS: ENQ. UUO

To request a lock on an ENQ. resource, use the ENQ. monitor call as follows:

```
MOVE    ac,[XWD fcncode,addr]   ;Set up call
ENQ.    ac,                     ;Enqueue the request
  error return                  ;Didn't get resource
normal return                   ;Got it
```

where **fcncode** is one of the four ENQ. functions described below; and **addr** is the address of the argument list. The argument list is described in Section 8.5.

The monitor attempts to lock each resource described by a lock block in the argument list.

The ENQ. function codes are:

Fcn-code	Symbol	Meaning
Ø	.ENQBL	Enqueue the request and wait until the requested resources are available.
1	.ENQAA	Dequeue the request immediately if any of the requested resources is unavailable.
2	.ENQSI	Enqueue the request and interrupt the program when the requested resources are available.
3	.ENQMA	Modify a previous request.

Each of these function codes and procedures are discussed below.

8.6.1 Requesting and Waiting for Locks

Your job can request locks on one or more resources and wait until the monitor grants the request. This type of request uses the ENQ. function code .ENQBL. When the request is enqueued, your job waits until the monitor can grant all the locks in the request.

8.6.2 Requesting Locks Only if Available

Your job can request locks on one or more resources, and prevent the monitor from queueing the request. That is, the monitor grants the request only if all the requested locks can be granted immediately; otherwise, the request is dequeued immediately. This procedure uses the ENQ. function code .ENQAA.

The monitor takes the error return, giving the ENQRU% error code, if any of the requested resources is unavailable. Otherwise, the monitor takes the normal return, having locked the requested resources.

8.6.3 Requesting and Interrupting when Locked

Your job can request locks on one or more resources, and have the monitor execute a software interrupt for the job when the requested resources are granted. This procedure uses the ENQ. function code .ENQSI.

If all the requested locks can be granted immediately, the monitor takes the normal return. If not, it takes the error return, giving the error code ENQRU%. The instruction at the error return should branch to some program task that can be performed while the job waits for the requested locks. This "task" could put the job to sleep.

When the monitor interrupts the job, your program should check the status word of the interrupt block. If the interrupt was caused by the granting of the requested resources, the request identifiers of the granted requests are inclusively ORed into the status word. If the interrupt was caused because the resource has been aborted, the sign bit of the status word will be on.

Your job must set up the PSI system for ENQ. interrupts before using the .ENQSI function of ENQ. (Refer to Chapter 6 for information on setting up the PSI system.)

8.6.4 Modifying a Previous Request

Your job can modify a previously queued request by using the ENQ. function .ENQMA. The function fails if you attempt to change a request from sharable to exclusive access when the resource already belongs to a sharer group.

If the requested modification is already in effect, the monitor makes no change and takes the normal return. If your job tries to modify a request that is not in the resource queue, the monitor takes the error return, giving the ENQNE% error code.

If your call to the .ENQMA function specifies more than one modification, and the monitor takes the error return, your job must use the ENQC. function .ENQCS to check the status of each request. The error code returned is only for the first error that occurred in the call.

8.7 DEQUEUEING REQUESTS: DEQ. UUO

To release a lock on an ENQ. resource (or cancel a request for the lock), use the DEQ. monitor call as follows:

```
MOVE    ac,[XWD fcncode,addr]   ;Set up call
DEQ.    ac,                     ;Dequeue the request
  error return                  ;Failed
normal return                   ;Dequeued
```

where **fcncode** is one of the function codes discussed below; and **addr** is the address of the argument list for function codes Ø and 1, and contains the request-id for function 2.

The monitor releases each resource described by a lock block in the argument list.

The DEQ. function codes are:

Fcn-code	Symbol	Meaning
Ø	.DEQDR	Dequeue or release a specified lock.
1	.DEQDA	DEQ. all requests and release all locks for the job.
2	.DEQID	DEQ. all requests and release all locks for a given request identifier.

Each of these DEQ. functions is discussed below.

8.7.1 Cancelling a Specific Request

Your job can cancel a specific request by using the DEQ. function .DEQDR. This function deletes the request from the queue or releases the lock on the requested resource.

If you use the .DEQDR function for a lock that is not in the queue and is not in force, the monitor takes the error return, giving the ENQNO% error code.

8.7.2 Cancelling All Requests for a Job

Your job can cancel all its ENQ. requests by using the DEQ. function .DEQDA. This function cancels all of your requests from the queue and releases all the locks you have placed on resources.

The monitor takes the error return, giving the ENQNO% error code, if you have no requests or locks.

8.7.3 Cancelling Requests Based on Request-id

Your job can cancel all its ENQ. requests that have the same request identifier by using the DEQ. function .DEQID. This function dequeues all requests that have the specified identifier, and releases all locks that were requested using the specified identifier.

The .DEQID function does not use an argument list; instead, it reads the request identifier from the field that would otherwise point to the address of the argument list. Thus, the calling sequence for the .DEQID function is:

```
MOVE    ac,[XWD .DEQID,request-id]
DEQ.    ac,
  error return
normal return
```

where request-id is the request identifier of the requests and locks that are to be cancelled.

If your job has no requests or locks for the given request-id, the monitor takes the error return, giving the ENQNO% error code.

8.8 CONTROLLING ENQ/DEQ: ENQC. UUO

The ENQ. facility offers the ENQC. monitor call for control of the facility itself. The calling sequence for the ENQC. UUO differs for each function code. The function codes for this monitor call are:

Fcn-code	Symbol	Meaning
Ø	.ENQCS	Obtain the status of a request.
1	.ENQCG	Obtain the ENQ. quota of a specified job.
2	.ENQCC	Set the ENQ. quota for a specified job.
3	.ENQCD	Examine the monitor's ENQ/DEQ database.

Note that some of these functions require privileges. The functions are described in the following sections.

8.8.1 Obtaining the Status of a Request

Your program can use an ENQC. function .ENQCS to check on the status of an ENQ. request, which is specified by the request identifier given with the request.

The calling sequence for the .ENQCS function is:

```
        MOVE    ac,[XWD .ENQCS,addr]
        MOVEI   ac+1,buffer
        ENQC.   ac,
          error return
        normal return
                .
                .
                .
buffer: BLOCK <lock blocks>*3
```

where addr is the address of the request block, as described in Section 8.8

On a normal return, the monitor returns a three-word status block for each request at buffer. Specify the amount of buffer space to reserve as number of lock blocks times 3. The format of the block returned at buffer is:

Word	Symbol	Contents
Ø	.ENQCF	Flags:

Bits	Symbol	Meaning
Ø	EQ.CFI	Invalid lock. The monitor sets this bit if it found an error in the corresponding lock request specification. When this bit is set, bits 18-35 contain an error code.
1	EQ.CFO	This user is owner.
2	EQ.CFQ	This user is in queue.
3	EQ.CFX	Owner's access is exclusive.
4-8		Reserved.
9-17	EQ.CFL	Level.

Word	Symbol	Contents		
		Bits	Symbol	Meaning
		18-35	EQ.CFJ	Job context handle or error code. (This is the same error code returned in the AC on an error return from ENQC.) If several jobs share the lock, this job context handle indicates only one of those sharers. However, if you are one of those sharers, this value will be your job context handle. It is possible that there may be no lock owner even though several jobs may have requested ownership. In this case, the monitor returns a -1 in bits 18-36.
1	.ENQCT			Time (in universal date-time format) that the lock was granted to its owner. This 36-bit value represents the last time someone was granted a request for the specified resource. This value is written in UDT format. If there is no owner of the resource, this word will be zero. This time stamp is useful if you want some type of 'watch dog' timer. Such a timer could periodically query the status of any queue in which throughput was of maximum importance. If it became obvious that the time stamp of the queue had not changed for a long time, the time process could signal the operator that someone had held the resource for longer than the 'allowable' time interval. The operator could then take whatever action was deemed appropriate.
2	.ENQCI			The left half (EQ.CIQ) contains the number of owners sharing the resource. The right half (EQ.CID) contains the request-id of the request. If you are currently in the queue for a specified resource, this value will be the request-id for your request in the queue (even if a lock has not yet been granted). However, if you are not the owner of the resource and not in the queue for the resource, the request-id will be that used by the owner of the resource. The request-id field allows the use of ENQ/DEQ with the PSI system.

8.8.2 Obtaining the Quota for a Job

You can use an ENQC. function (.ENQCG) to obtain the ENQ. quota for a user. The calling sequence for this function is:

```
        MOVE   ac,[XWD .ENQCG,addr]
        ENQC.  ac,
          error return
        normal return
               .
               .
               .
addr:   XWD Ø,jobno
```

where addr is the address of the argument list; and jobno is the number of the job whose quota is required (-1 for your own job).

The monitor returns the quota for the specified job in ac.

8.8.3 Setting the Quota for a Job

If you have the required privileges (JP.POK, JACCT, or the [1,2] PPN), you can use an ENQC. function (.ENQCC) to set the ENQ. quota for a user. The calling sequence for this function is:

```
        MOVE   ac,[XWD .ENQCC,addr]
        ENQC.  ac,
          error return
        normal return
               .
               .
               .
addr:   XWD quota,jobno
```

where addr is the address of the argument list; quota is the new quota to be set for the job; and jobno is the number of the job whose quota is to be set (-1 for your own job).

The monitor sets the quota for the specified job.

8.8.4 Dumping the ENQ. Database

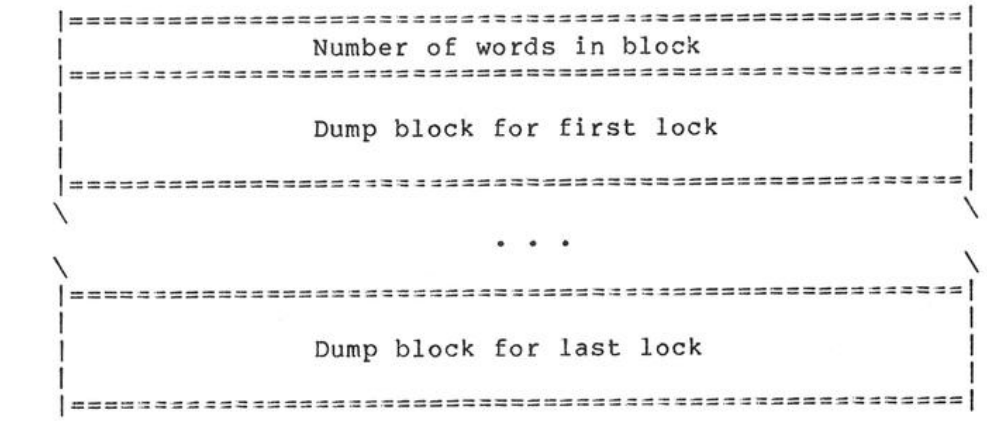
If you have the required privileges (JP.SPM, JP.SPA, JACCT, or [1,2]), you can use an ENQC. function (.ENQCD) to dump the monitor's ENQ. database. The calling sequence for this function is:

```
        MOVE   ac,[XWD .ENQCD,buffer]
        ENQC.  ac,
          error return
        normal return
               .
               .
               .
buffer: XWD Ø,buflength
        BLOCK  buflength
```

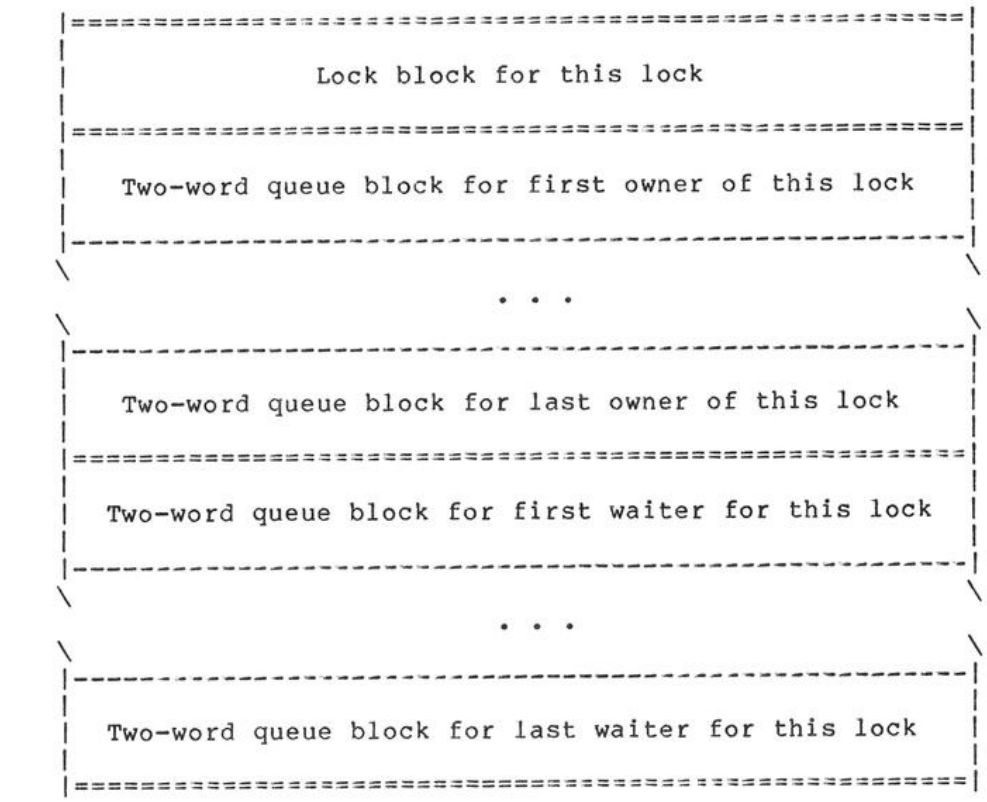
where buffer is the address of a buffer for returned data; and buflength is the length of the buffer.

The monitor returns a dump of the ENQ. database beginning at buffer. If the database does not fill the entire buffer, the balance is filled with zeros; if the database will not fit into the buffer, it is truncated.

The following diagram shows the general format of the ENQC. dump:



The format for each dump block is:



where each lock block is in the format:

Word	Symbol	Contents
Ø	.EQDFL	Flags, level, and lock identifier:

Bits	Symbol	Meaning
Ø	EQ.DLB	This is a lock block.
2	EQ.DLT	This lock has text.
5	EQ.DLN	This lock block will not be dequeued on a reset.
6	EQ.DLA	This lock is aborted; no new requests will be granted.
9-17	EQ.DFL	Level number.
18-35	EQ.DFI	Lock identifier. (That is, the access table address or code of -1, -2, or -3.)

The flag word (.EQDFL) is returned as -1 at the end of the list of queue blocks.

Word	Symbol	Contents
1	.EQDPR	Pooled request counts:

Bits	Symbol	Meaning
Ø-17	EQ.DPS	Size of pool.
18-35	EQ.DPL	Number of resources available in the pool.

Word	Symbol	Contents
2	.EQDTS	Time stamp.
3	.EQDSU	Resource identification string or numeric code.

The format of each 2-word queue block is:

Word	Symbol	Contents
Ø	.EQDFL	Flags in the left half, and associated job number in the right half.

Bits	Symbol	Meaning
1	EQ.DLO	This is the lock owner.
3	EQ.DXA	Exclusive access.
4	EQ.DJW	This job is blocked waiting for lock.
7	EQ.DQI	This queue block is invisible.
8	EQ.DQD	This queue block will be checked for deadlocks.

Word	Symbol	Contents
1	.EQDGI	Group number and request identifier:

Bits	Symbol	Meaning
Ø-17	EQ.DGR	Group or number requested.
18-35	EQ.DRI	Request identifier.

8.9 ENQ. ERRORS

For errors from ENQ. monitor calls (ENQ., DEQ., and ENQC.), the monitor returns one of the following error codes in the AC:

Code	Symbol	Error
1	ENQRU%	At least one of the requested resources is not available.
2	ENQBP%	You requested an illegal number of pooled resources.
3	ENQBJ%	You specified an illegal job number.
4	ENQBB%	You specified an illegal byte size for the byte pointer. The byte pointer must be 1 to 36 (decimal).
5	ENQST%	ASCIZ string is too long. It must be less than 150 (decimal) characters.
6	ENQBF%	You specified an illegal function code.
7	ENQBL%	You specified an illegal argument list length. The total length must be the header-block-size plus (request-block-size times number-of-requests).
10	ENQIC%	You specified an illegal number of requests.
11	ENQBC%	You specified an illegal channel number.
12	ENQPI%	Your program does not have enough privileges for specified function.
13	ENQNC%	Not enough core available, or the maximum number of active locks (item %EQMAQ in GETTAB table .GTENQ) has been exceeded.
14	ENQFN%	File is not open or device is not a disk.
15	ENQIN%	Address for byte pointer cannot be indirect or indexed.
16	ENQNO%	Your program cannot dequeue resources not owned by your job.
17	ENQLS%	Levels not specified in ascending order.
20	ENQCC%	Illegal modification of ownership; you cannot change a request from shared to exclusive ownership. DEQ. first, then re-ENQ.
21	ENQQE%	Enqueue quota exceeded; your quota is set by the system administrator.
22	ENQPD%	Number of resources in pool disagrees with number in your request.
23	ENQDR%	Duplicate request; the request is identical to a request already queued by your job.
24	ENQNE%	The resource cannot be dequeued because it is not enqueued for your job.
25	ENQLD%	Level in request does not match lock.
26	ENQED%	Not enough privileges for ENQ/DEQ; you must have JP.ENQ set.
27	ENQME%	Mask too long or lengths do not match.
30	ENQTE%	Lock-associated block is too long.
31	ENQAB%	A resource that was requested has been marked as aborted.
32	ENQGF%	An eternal lock cannot be placed on a "ghost file" (that is, a file that is in the process of being created or superseded on disk).
33	ENQDD%	A deadlock was detected.
34	ENQTL%	The specified time limit was exceeded.

8.10 EXAMPLE USING THE ENQ. FACILITY

An example of the ENQ. monitor call is shown below.

```
TITLE     ENQEXA - A simple ENQ application
SUBTTL    PETER VATNE 6-JUN-83

          SEARCH JOBDAT,MACTEN,UUOSYM ;Get standard symbol definitions
          SALL                        ;Make listing pretty

          T1=1                        ;Temporary ACs
          T2=2
          T3=3
          T4=4
          P=17                        ;Push down pointer

          IO==1                       ;Input-output disk channel

          LN$PDL==40                  ;Stack size
          LN$DAT==200                 ;Data block size

DEFINE    ERROR  (COD,TXT),<          ;;Standard error macro
          JRST   [OUTSTR [ASCIZ/?ENQ'COD 'TXT/]
                 RESET                ;;Clear all locks, etc.
                 MONRT.               ;;Return to monitor fast
                 JRST START]          ;;Start all over again>

          TWOSEG 400000               ;Make program sharable

START:    JFCL                        ;In case of CCL entry
          RESET                       ;Reset the world
          MOVE   P,[IOWD LN$PDL,PDLST] ;Initialize push down list
          MOVE   T1,[SIXBIT /DATA/]
          MOVEM  T1,LKPBLK            ;File name is DATA
          MOVSI  T1,'DAT'
          MOVEM  T1,LKPBLK+1          ;File extension is .DAT
          MOVE   T1,[6,,[IO,,.FOMAU   ;Channel=IO,mode=MULTI-USER
                 .IODMP               ;Mode=DUMP
                 SIXBIT /DSK/         ;Device=DSK
                 0,,0                 ;No buffer headers
                 0,,0                 ;No buffers
                 ENTBLK,,LKPBLK]]     ;File name blocks
          FILOP. T1,                  ;Open a file in simul update mode
           ERROR (COF,Can't open file DATA.DAT for simultaneous update)

RECORD:   MOVE   T2,[POINT 7,MSGBLK]  ;Point to start of message
          MOVEI  T3,LN$DAT*5          ;Count of character
          SETZM  MSGBLK               ;Zero out message block
          MOVE   T1,[MSGBLK,,MSGBLK+1] ;..
          BLT    T1,MSGBLK+LN$DAT-1   ;..
          OUTSTR [ASCIZ /INPUT>/];PROMPT
```

```
TTYINP:  INCHWL T1                      ;Read a character (line mode)
         CAIN   T1,.CHCNZ               ;^Z?
          EXIT                          ;Yes, done with program
         SOSL   T3                      ;Room left in message block
          IDPB  T1,T2                   ;Yes, append to message
         CAIE   T1,.CHLFD               ;End of line?
          JRST  TTYINP                  ;No, loop for more

         MOVE   T1,[.ENQBL,,[1B5+1B17+3 ;Header=1,locks=1,length=3
                EQ.FCW+IO               ;Code word,level=Ø,channel=1
                -1]]                    ;Arbitrary code
         ENQ.   T1,                     ;Lock the resource
          ERROR (CEB,Can't ENQ. a block for DATA.DAT)
         USETI  T1,1                    ;Set to block 1
         INPUT  T1,[IOWD LN$DAT,DATBLK
                Z]                      ;Read the block
         USETO  T1,1                    ;Set to block 1
         OUTPUT T1,[IOWD LN$DAT,MSGBLK
                Z]                      ;Write new message
         MOVE   T1,[.DEQDA,,Ø]          ;DEQ all locks
         DEQ.   T1,                     ;Unlock the resource
          ERROR (CDB,Can't DEQ. a block for DATA.DAT)
         OUTSTR [ASCIZ /              => /]
         OUTSTR DATBLK                  ;Type the contents of the block
         JRST   RECORD                  ;Do it again

         LIT                            ;Store the literals in the high seg

         RELOC                          ;Switch to low segment storage

LKPBLK:  BLOCK  4                       ;Storage for LOOKUP block
ENTBLK:  BLOCK  4                       ;Storage for ENTER block
DATBLK:  BLOCK  LN$DAT                  ;Storage for data block
MSGBLK:  BLOCK  LN$DAT                  ;Storage for message input
PDLST:   BLOCK  LN$PDL                  ;Storage for push-down list

         END    START
```

CHAPTER 9

PROGRAMMING FOR REALTIME EXECUTION

Most jobs run under the TOPS-10 monitor as timesharing jobs. The system scheduler schedules them for CPU time. (Refer to the SCHED. monitor call in Volume 2.) But realtime jobs are assigned higher priorities than timesharing jobs and the CPU serves them on an event-driven basis. Timesharing jobs use the CPU time left after realtime jobs have been served.

NOTE

Realtime execution is not available on KS processor systems.

Realtime programs connect devices to the CPU priority interrupt (PI) system. The device is then under exclusive control of the realtime program, and is called a realtime device. Refer to the DECsystem 10/20 Processor Reference Manual for a discussion of the hardware Priority Interrupt system.

A realtime program must be locked in core so that the program can service the device at interrupt level. (The system cannot do paging/swapping at interrupt level.) A realtime program cannot connect to a device unless the program is locked in core.

The monitor calls that are most important for realtime programs are:

RTTRP Connects and releases realtime devices.

UJEN Dismisses a realtime interrupt.

HPQ Places a job in a high-priority run queue. You should not place a job in a high-priority run queue if the job will run for a long time. Doing so can cause other jobs to lose data.

TRPSET Temporarily suspends execution of all other jobs to facilitate realtime processing.

LOCK Locks a job in core.

9.1 CONNECTING REALTIME DEVICES

To connect a realtime device, use the RTTRP monitor call. A realtime device is controlled in one of four ways; this control is specified in the RTTRP call that connects the device. The four control modes are:

- Block mode, in which the monitor reads an entire block of data before it interrupts your program. To accomplish this, the monitor places BLKI/BLKO in the skip chain.

- **Fast block mode**, in which the monitor also reads an entire block of data before it runs the interrupt program, but responds faster than normal block mode. To accomplish this, the monitor places BLKI/BLKO at the interrupt location.

- **Single mode**, in which the monitor runs your interrupt routine each time the device interrupts. To accomplish this, the monitor places an XPCW instruction at the interrupt location.

- **Executive Process Table (EPT) relative mode**, in which the monitor responds to a vectored interrupt. To accomplish this, the monitor places the XPCW instruction into the EPT vector.

Note that EPT mode responds to vectored interrupts. The first three modes are for non-vectored interrupts. Your choice of the first three modes should depend on the speed of response required by your program.

Interrupts are distinguished as standard or vectored, depending on how they get a new PC. With standard interrupts, two memory locations are assigned to each channel starting at locations 40 + 2n and 41 + 2n in the EPT. The processor starts an interrupt on channel n by executing the instruction in location 40 + 2n in the EPT. The standard interrupt follows the CONSO skip chain testing all the devices in the following way:

```
40+2n:     XPCW CH'N

CH'N:      Old flags
           Old PC
           New flags
           New PC (always .+1)
           JRST to CONSO skip chain

DEV'INT:   CONSO DEV1,bits
           JRST DEV2'INT
           .
           .
           .
           DISMISS

DEV2'INT:  CONSO DEV2,bits
           JRST DEV3'INT
           .
           .
           .
           DISMISS

           The last entry on the chain contains an XJEN CH'N
```

With vectored interrupts, the device gives the new PC to the hardware by using an offset into the Executive Process Table. This location contains the instruction which the hardware executes on an interrupt. In the following example, the interval time uses location 514.

```
EPT+514/     XPCW TM0INT
```

The interval timer is a vectored interrupt device, as are the RH20's.

Normally, the interrupt service routine is run in user mode with user I/O privileges. However, your program can also use executive mode for its realtime interrupts if it requires extremely fast response. (See Section 9.1.4.)

Using normal or fast block mode places a BLKI or BLKO instruction directly on a PI level. Since this instruction is executed in executive mode, you must first lock your job in executive virtual memory.

In normal block mode, the monitor places the BLKI or BLKO instruction directly after the entry for the device in the monitor's CONSO skip chain. Any number of realtime devices using either single mode or normal block mode can be placed on any available PI level. The level is considered available if it is not used for a BLKI or BLKO instruction by the system or by other realtime users, including the APR channel. (The average overhead per device on the same PI level is 5.5 microseconds per interrupt.)

In fast block mode, the monitor places the BLKI or BLKO instruction directly in the PI location. This requires that the interrupt level be dedicated to the realtime job during data transfers.

In single mode, the monitor places an XPCW instruction into the interrupt location to pass control to your program on every interrupt.

In EPT-relative mode, the monitor places the XPCW into the EPT vector to pass the control to your program on an interrupt at that vector.

On a normal return from a RTTRP monitor call, the job is granted IOT privileges, allowing the job to execute restricted I/O instructions from user mode. These instructions could halt the system if used improperly. The restricted instructions are:

CONO	DATAI	JEN
CONI	DATAI	XSFM
CONSO	BLKI	XJEN
CONSZ	BLKØ	

A RTTRP monitor call with the PI level (see below) given as zero produces the IOT privilege without setting up a device.

A realtime device must be placed on the same interrupt level in its RTTRP monitor call, and in the CONO instruction which establishes the interrupt level for the device.

If a CONSO bit mask is set up for a device, but the device has not been placed on its proper interrupt level, and if a flag for the device is on, an interrupt to your program can occur. This is because there is a CONSO skip chain for each interrupt level; if a device interrupts whose CONSO instruction is further down the chain than the device itself, the CONSO is executed. If a hardware device flag is set (and the corresponding bit is set in the CONSO bit mask), the CONSO skips and an interrupt to your interrupt service routine occurs even though the device did not interrupt.

To avoid this, your program can store the CONSO bit mask in the your area (for example, at MSKADD) and use an indirect address in the CONSO instruction (for example, CONSO device, @MSKADD). This allows your program to chain the device to its interrupt level, but keep a zero bit mask until the device is placed on its interrupt level by a CONO instruction.

When your program removes a device from the interrupt chain, it must also remove it from the interrupt level with a CONO device,Ø instruction.

The calling sequence for the RTTRP monitor call is:

```
MOVEI  ac,addr
RTTRP  ac,
  error return
normal return
```

where addr is the address of the argument list. The contents of the argument list depend on which mode is being set up by the call. The argument list for each mode is discussed in Sections 9.1.1 through 9.1.5.

The RTTRP functions require the following types of information:

- o Realtime interrupt levels

  The realtime interrupt level (given as **level** in the argument lists in this chapter) is the PI level for the realtime device. You may specify any of the following in the PI level field:

  1. A positive PI channel. Specifying level Ø with the RTTRP monitor call releases the device. Levels 1 and 2 are often reserved for tape devices such as TD1Ø or TM1Ø. Levels 3 through 6 are legal levels. Level 7 is always reserved for the system.

     If you lock your job on a CPU, and then specify a positive PI channel, the system assumes the realtime device is on that CPU. If your job is not locked on a CPU, the system locks it on CPUØ, and attaches the device to the channel you indicated. The system then removes all occurrences of the device on any PI channel on the same CPU in the system. This is useful when a device uses multiple PI levels for flags and data or a software-generated interrupt is used for lower-level processing.

  2. A negative PI channel. When your job is locked on a CPU, the system assigns the PI channel in the same manner as for a positive PI channel specification, but does not remove occurrences of the device from other PI channels. When your job is not locked on a CPU, and you assign a negative PI channel, the system attaches the device to the channel the same way as for a positive specification, but does not remove other occurrences of the device.

  3. A bit mask in the form:

     1B Ø + <CPU no.>B8 + <PI chan.>B17.

     The system assigns the device on the specified channel on the CPU you indicate in the CPU number (CPU no.) field. If you do not specify a CPU, the system assigns the device to the channel as though you specified a positive PI channel.

4. A bit mask in the form:

   1B0 + 1B1 + <CPU no.>B8 + <PI chan.>B17.

   The system assigns the device on the specified channel on the CPU you indicate in the CPU number (CPU no.) field. If you do not specify a CPU, the system assigns the device to the channel as though you specified a negative PI channel.

- Trap location

  The location to which a realtime interrupt traps is given as **realtrap** in the argument lists in this chapter. Realtrap is the start of the interrupt service routine. When a user mode realtime interrupt occurs, the monitor saves all accumulators; therefore a realtime interrupt service routine can overwrite them without loss.

- APR trap location

  The location to which APR conditions trap is given as **aprtrap** in the argument lists in this chapter. On an APR trap such as NXM or PDL overflow, the monitor executes a JSR instruction to the trap address. (If a realtime device is on a higher interrupt level than the APR level, no APR conditions on the device are detected.)

- Device name

  A **realtime device** is defined by a symbol for the device code. See the MACRO Reference Manual for a list of I/O device symbols.

- Bit mask

  A **mask** is defined in each CONSO instruction in a realtime trap. The mask bits differ for each device. See the Hardware Reference Manual for a list of the bits in each mask. The CONSO instruction can be written as:

```
        CONSO     device,mask
```

  Where the mask is in the address field of the instruction; or the CONSO instruction can be written as:

```
        CONSO     device,@mskadd
        . . .
mskadd: XWD       0,mask
```

  Where **mskadd** is the address of the word (in your area, which is locked in executive virtual memory) containing the mask; and **mask** is the 18-bit mask value.

  The indirect addressing of this mask allows your program to store the mask in your memory area. However, the word containing the mask must not have the indirect or index fields set (that is, Bits 13 through 17 must be 0).

- BLKI/BLKO pointer word

  The address (in your memory area) of the BLKI/BLKO pointer word is given in the argument lists in this chapter as **blockaddr**. On a realtime interrupt, the monitor adds a relocation constant to this pointer; therefore your program must restore the pointer to its original value after each interrupt.

- Interrupt flags

  Realtime interrupt flags (given as **flags** in the argument lists in this chapter) are as follows:

Flag	Meaning
Ø	If set, Bits 6-8 contain the CPU number of the CPU to which the device is connected. If this flag is not set, the device is on the same CPU that the program is locked on. If the program is not locked on this CPU, RTTRP locks it on CPUØ:.
1	There are multiple device references in the interrupt chain. Thus, the device can generate interrupts on more than one PI channel.
15	This is an EPT-mode interrupt; the address given in **addr+2** is an offset into the executive page table.
16	The device indicated in the argument list supports and produces vectored interrupts.
17	The monitor patches the trap to a JSR to a trap address. If this bit is not set, the monitor patches the trap to a JSR to the context switcher. If this bit is set, your interrupt service routine is entered in executive mode with no accumulators saved. (Refer to Section 9.1.4.)

Should the job running realtime abort the program using CTRL/C, the monitor uses the device field to execute a CONO instruction to reset the device and remove it from realtime processing.

9.1.1 Normal Block Mode

The RTTRP argument list for normal block mode is:

```
addr:   XWD     level,realtrap
        XWD     flags,aprtrap
        CONSO   device,mask
          BLKx  device,blockaddr
```

Where **level** is the interrupt level for the device; **realtrap** is the address of the interrupt service routine for the given device; **flags** are realtime interrupt flags; **aprtrap** is the trap location for all APR traps; **BLKx** is your choice of BLKI or BLKO; **device** is the device code for the realtime device; **mask** is the bit mask; and **blockaddr** is the address in your area of the BLKI/BLKO pointer word.

Example

```
TITLE   RTNBLK - Papertape read test in BLKI mode

SEARCH  UUOSYM                          ;Standard symbols

;Some values

TAPE=400                                ;No more tape in reader if TAPE=0
BUSY=20                                 ;Device is busy reading
DONE=10                                 ;A character has been read
AC1=1                                   ;Work register
AC2=2                                   ;Work register
TABLEN=200                              ;Table length
PICHAN=6                                ;PI channel

;Storage

;Realtime data block

RTBLK:  XWD     PICHAN,TRPADR           ;PI channel and trap address
        XWD     0,APRTRP                ;Flags and APR trap
        CONSO   PTR,DONE                ;Wait only for done flag
        BLKI    PTR,POINTR              ;Read a block at a time

POINTR: IOWD    TABLEN,TABLE            ;Pointer for BLKI instruction
OPOINT: BLOCK   1                       ;Original pointer
TABLE:  BLOCK   TABLEN                  ;Table area for data being read
DONFLG: BLOCK   1                       ;PI level to user level command

RTBLK1: EXP 0                           ;Data block to remove PTR
        EXP 0                           ; from PI channel
        CONSO   PTR,0
        EXP     0

                                        ;Here we begin

BLKTST: RESET                           ;Reset the program
        MOVE    AC1,[LK.HLS+LK.LLS]     ;Lock both segments
        LOCK    AC1,                    ;Lock the program in core
          JRST  FAILED                  ;LOCK call failed
        MOVEI   AC1,RTBLK1              ;Get address of realtime block
        RTTRP   AC1,                    ;Get user IOT privilege

          JRST  FAILED                  ;RTTRP call failed
        CONO    PTR,0                   ;Clear all PTR bits
        SETZM   DONFLG                  ;Initialize the done flag
        MOVEI   AC1,RTBLK               ;Get address of realtime data block
        RTTRP   AC1,                    ;Put realtime device on PI level
          JRST  FAILED                  ;RTTRP call failed
        MOVE    AC1,POINTR              ;Get relocated pointer for later
        MOVEM   AC1,OPOINT              ;Store for interrupt level use
        HLRZ    AC2,RTBLK               ;Get PI number from RTBLK
        TRO     AC2,BUSY                ;Set up CONO bits to start tape
        CONO    PTR,(AC2)               ;Turn on PTR
        MOVEI   AC1,5                   ;For five seconds . . .
        SLEEP   AC1,                    ;Sleep
        SKIPN   DONFLG                  ;Finished reading the tape?
        JRST    .-3                     ;No, back to sleep
        EXIT                            ;Yes, stop the job

                                        ;Here's the trap
```

```
TRPADR: CONSO   PTR,TAPE            ;End-of-tape?
        JRST    TDONE               ;Yes, stop the job
        MOVE    AC1,OPOINT          ;No, get original pointer
        MOVEM   AC1,POINTR          ;Store in pointer location
        UJEN                        ;Dismiss the interrupt

                                    ;Here on end-of-tape

APRTRP: BLOCK   1                   ;APR error trap address

TDONE:  MOVEI   AC1,RTBLK1          ;Set up to remove PTR
        CONO    PTR,0               ;Take device off hardware PI level
        RTTRP   AC1,                ;Take device off software PI level
          JFCL                      ;Ignore errors
        SETOM   DONFLG              ;Indicate that reading is over
        UJEN                        ;Dismiss the interrupt

                                    ;Here if LOCK or RTTRP call failed

FAILED: OUTSTR  [ASCIZ /RTTRP or LOCK call failed.
/]
        EXIT                        ;Stop the job

        END     BLKTST
```

9.1.2 Fast Block Mode

The RTTRP argument list for fast block mode is as follows:

```
     addr:      XWD     level,realtrap
                XWD     flags,aprtrap
                  BLKx  device,blockaddr
```

where **level** is the interrupt level for the device; **realtrap** is the address of the interrupt service routine for the given device; **flags** are realtime interrupt flags; **aprtrap** is the trap location for all APR traps; **BLKx** is your choice of BLKI or BLKO; **device** is the device code for the realtime device; and **blockaddr** is the address in your area of the BLKI/BLKO pointer word.

Example

```
TITLE   RTFBLK - Papertape read test in BLKI mode

SEARCH  UUOSYM                      ;Standard symbols

;Some values

TAPE=400                            ;No more tape in reader
BUSY=20                             ;Device is busy reading
DONE=10                             ;A character has been read
TABLEN=5                            ;Length of table
AC1=1                               ;Work register
AC2=2                               ;Work register
PICHAN=6                            ;PI channel
```

```
;Storage

POINTR: IOWD    TABLEN,TABLE        ;Pointer for BLKI instruction
OPOINT: BLOCK   1                   ;Original pointer word for BLKI
TABLE:  BLOCK   TABLEN              ;Table area for data being read
DONFLG: BLOCK   1                   ;PI level to user level command

RTBLK1: BLOCK   1                   ;Data block to remove PTR
        BLOCK   1                   ; from PI channel
        CONSO   PTR,Ø
        BLOCK   1

;Realtime data block

RTBLK:  XWD     PICHAN,TRPADR       ;PI channel and trap address
        XWD     Ø,APRTRP            ;APR error trap address
        BLKI    PTR,POINTR          ;Read a block at a time
        BLOCK   1

;Here we begin

BLKTST: RESET                       ;Reset the program
        MOVE    AC1,[LK.HLS+LK.LLS];Lock both segments
        LOCK    AC1,                ;Lock the job in core
          JRST  FAILED              ;LOCK call failed
        SETZM   DONFLG              ;Initialize done flag
        MOVEI   AC2,RTBLK           ;Get address of realtime data block
        RTTRP   AC2,                ;Put realtime device on PI level
          JRST  FAILED              ;RTTRP call failed
        MOVE    AC2,POINTR
        MOVEM   AC2,OPOINT
        HLRZ    AC2,RTBLK           ;Get PI number from RTBLK
        TRO     AC2,BUSY            ;Set up CONO bits to start tape
        CONO    PTR,(AC2)           ;Turn on PTR
        MOVEI   AC1,5               ;For five seconds . . .
        SLEEP   AC1,                ;Sleep
          SKIPN DONFLG              ;Finished reading the tape?
        JRST    .-3                 ;No, back to sleep
        EXIT                        ;Yes, stop the job

;Here's the trap

TRPADR: CONSO   PTR,TAPE            ;End-of-tape?
        JRST    TDONE               ;Yes, stop the job
        MOVE    AC2,OPOINT          ;Get original pointer
        MOVEM   AC2,POINTR          ;Restore BLKI pointer
        UJEN                        ;Dismiss the interrupt

;Here on end-of-tape

APRTRP: BLOCK   1                   ;APR trap address

TDONE:  MOVEI   AC2,RTBLK1          ;Set up to remove PTR
        CONO    PTR,Ø               ;Take device off hardware PI level
        RTTRP   AC2,                ;Take device off software PI level
          JFCL                      ;Ignore errors
        SETOM   DONFLG              ;Indicate that reading is done
        UJEN                        ;Dismiss the interrupt
```

```
;Here on failure for LOCK or RTTRP call

FAILED: OUTSTR  [ASCIZ /RTTRP or LOCK call failed.
/]
        EXIT                            ;Stop the job

        END     BLKTST
```

9.1.3 Single Mode

The RTTRP argument list for single mode is as follows:

```
     addr:      XWD       level,realtrap
                XWD       flags,aprtrap
                CONSO     device,mask
                BLOCK     1
```

Where **level** is the interrupt level for the device; **realtrap** is the address of the interrupt service routine for the given device; **flags** are realtime interrupt flags; **aprtrap** is the trap location for all APR traps; **device** is the device code for the realtime device; and **mask** is an 18-bit mask for the CONSO instruction.

Example

```
TITLE   RTSNGL - Papertape read test using CONSO chain

SEARCH  UUOSYM                          ;Standard symbols

;Some values

PIOFF=400                               ;Turn PI system off
PION=200                                ;Turn PI system on
TAPE=400                                ;No more tape in reader
BUSY=20                                 ;Device is busy reading
DONE=10                                 ;A character has been read
PICHAN=5                                ;PI channel
AC1=1                                   ;Work register
AC2=2                                   ;Work register

;Storage

PDATA:  BLOCK   1                       ;Read data into this location

;Realtime data block

RTBLK:  XWD     PICHAN,TRPADR           ;PI channel and trap address
        XWD     0,APRTRP                ;APR error trap address
        CONSO   PTR,@PTRCSO             ;Indirect CONSO bit mask = PTRCSO
        BLOCK   1                       ;No BLKI/O instruction

PTRCSO: BLOCK   1                       ;CONSO bit mask

DONFLG: BLOCK   1                       ;PI level to user level command

RTBLK1: BLOCK   1                       ;Data block to remove PTR
        BLOCK   1                       ; from PI channel
        CONSO   PTR,0
        BLOCK   1
```

```
;Here we begin

PTRTST: RESET                          ;Reset the program
        MOVE    AC1,[LK.HLS+LK.LLS];Lock both segments
        LOCK    AC1,
          JRST  FAILED                 ;LOCK call failed
        SETZM   PTRCSO                 ;Zero CONSO bits
        SETZM   DONFLG                 ;Initialize done flag
        MOVEI   AC1,RTBLK              ;Address of realtime data block
        RTTRP   AC1,                   ;Put realtime device on PI level
          JRST  FAILED                 ;RTTRP call failed
        MOVEI   AC1,DONE               ;Set up CONSO bit mask
        HLRZ    AC2,RTBLK              ;Get PI number from RTBLK
        TRO     AC2,BUSY               ;Set up CONO bits to start tape
        CONO    PI,PIOFF               ;Guard against interrupts
        MOVEM   AC1,PTRCSO             ;Store CONSO bit mask
        CONO    PTR,(AC2)              ;Turn on PTR
        CONO    PI,PION                ;Allow interrupts again
        MOVEI   AC1,5                  ;For five seconds . . .
        SLEEP   AC1,                   ;SLEEP
          SKIPN DONFLG                 ;Finished reading the tape?
        JRST    .-3                    ;No, back to sleep
        EXIT                           ;Finished

;Here's the trap

TRPADR: CONSO   PTR,TAPE               ;End of tape?
        JRST    TDONE                  ;Yes, stop job
        DATAI   PTR,PDATA              ;Read in data word
        UJEN                           ;Dismiss the interrupt

;Here on end-of-tape

APRTRP: BLOCK   1                      ;APR trap address

TDONE:  MOVEI   AC1,RTBLK1             ;Set up to remove PTR
        CONO    PTR,Ø                  ;Take device off hardware PI level
        RTTRP   AC1,                   ;Take device off software PI level
          JFCL                         ;Ignore errors
        SETOM   DONFLG                 ;Indicate that read is done
        SETZM   PTRCSO                 ;Clear CONSO bit mask
        UJEN                           ;Dismiss the interrupt

;Here on failure of RTTRP or LOCK call

FAILED: OUTSTR  [ASCIZ /RTTRP or LOCK call failed.
/]
        EXIT                           ;Stop the job

        END     PTRTST
```

9.1.4 EPT Mode

Executive page table (EPT) mode addresses an interrupt by an offset into the executive page table. This mode is only for KL1Ø processors.

The RTTRP argument list for EPT mode is as follows:

```
addr:   XWD     level,realtrap
        XWD     flags,aprtrap
        EXP     <device>B9+evaddr
        BLOCK   1
```

Where **level** is the interrupt level for the device; **realtrap** is the address of the interrupt service routine for the given device; **flags** are realtime interrupt flags; **aprtrap** is the trap location for all APR traps; **device** is the device code for the realtime device; and **evaddr** is an offset within the executive process table of an EPT-mode interrupt vector.

9.1.5 Exec-Mode Trapping

In special cases, the realtime user requires a faster response time than that offered by the RTTRP monitor call when executed in user mode. To accommodate these cases, you can specify a special status bit in the RTTRP monitor call that gives the program control in exec mode. Exec-mode trapping gives response times of less than 1Ø microseconds to real-time interrupts. To use this exec-mode trapping feature, the job must have real-time privileges (granted by LOGIN) and be locked in core (accomplished by using the LOCK monitor call). The job must also be mapped contiguously in exec virtual memory. The privilege bits that must be set in JBTPRV are:

JP.TRP(Bit 15)
JP.LCK(Bit 14)
JP.RTT(Bit 13)

Several restrictions must be placed on user programs in order to achieve this level of response. On receipt of an interrupt, program control is transferred to the user's real-time program without saving the accumulators and with the processor in exec mode. Therefore, the user program must save and restore all accumulators that are used, must not execute any monitor calls, and cannot leave exec mode. This means that the programs must be self-relocating (that is, through the use of an index or base register).

CAUTION

> Improper use of the exec mode feature of the RTTRP monitor call can cause the system to fail in a number of ways. Unlike the user mode feature of RTTRP, errors are not prevented because, in exec mode, the programs run with no accumulators being saved.

To specify RTTRP exec mode trapping, Bit 17 of the second word in the RTTRP argument block must be set to 1. This implies that no context switching is to be done and that a JSR instruction is to be used to enter the user's realtime interrupt routine. The user program must save and restore all accumulators and should dismiss the interrupt with a JRSTF to an indirect location. This instruction must be set up prior to the start of the real-time device as an absolute or unrelocated instruction. This can be done because the LOCK monitor call returns the exec virtual page numbers of the low and high segments after the job is locked in a fixed place in memory.

The exec mode trapping feature can be used with any of the standard forms of the RTTRP monitor call.

Example

```
TITLE   RTFXEX - Realtime trapping in executive mode

SEARCH  UUOSYM                          ;Standard symbols

;Some values

TAPE=400                                ;No more tape in reader if TAPE=0
BUSY=20                                 ;Device is busy reading
DONE=10                                 ;A character has been read
I=1                                     ;Work register
AC2=2                                   ;Work register
PICHAN=6                                ;PI channel

;Storage

DONFLG: BLOCK   1                       ;PI level to user level command

APRTRP: BLOCK   1                       ;APR trap address

;Realtime data block

RTBLK:  XWD     PICHAN,TRPADR           ;PI channel and trap address
        XWD     1,APRTRP                ;Bit 17 says trap in exec mode
        CONSO   PTR,DONE
        BLOCK   1

PDATA:  BLOCK   1                       ;Data word
INDEX:  BLOCK   1                       ;Base index register

;Here we begin

RTEXEC: RESET                           ;Reset the program
        SETZM   DONFLG                  ;Initialize the done flag
        MOVE    AC2,[LK.HLS+LK.LLS]     ;Lock both segments
        LOCK    AC2,                    ;Lock the job in core
                                        ; (Absolute address of job
                                        ; returned in AC2)
          JRST  FAILED                  ;LOCK call failed
        HRRZS   AC2,                    ;Get only low-segment address
        LSH     AC2,9                   ;Justify the address
        MOVEM   AC2,INDEX               ;Save base address for later
        ADDM    AC2,EXCHWD              ;Relocate interrupt level program
        ADDM    AC2,JENWD               ;Relocate interrupt instruction
        MOVEI   AC2,RTBLK               ;Connect realtime device
        RTTRP   AC2,                    ; to the PI system
          JRST  FAILED                  ;RTTRP call failed
        CONO    PTR,20+PICHAN           ;Start realtime device reading
SLEEP:  MOVEI   AC2,^D1000              ;For 1000 microseconds . . .
        HIBER   AC2,                    ;Sleep
          JRST  FAILED                  ;HIBER call failed
        SKIPN   DONFLG                  ;Interrupt level program done?
        JRST    SLEEP                   ;No, back to sleep
        EXIT                            ;Yes, stop the job
```

```
TRPADR: BLOCK   1                       ;JSR TRPADR on interrupt
EXCHWD: EXCH    I,INDEX                 ;Set up index register
        CONSO   PTR,TAPE                ;Tape finished?
        JRST    TDONE(I)                ;Yes, stop the reader
        DATAI   PTR,PDATA(I)            ;No, read next character
RETURN: EXCH    I,INDEX(I)              ;Restore ACs used
JENWD:  JRSTF   @TRPADR                 ;Dismiss the interrupt

TDONE:  CONO    PTR,Ø                   ;Take the reader off-line
        SETOM   DONFLG(I)               ;Indicate that the tape is finished
        JRST    RETURN(I)               ;Go dismiss the interrupt

FAILED: OUTSTR  [ASCIZ /LOCK, RTTRP, or HIBER call failed.
/]
        EXIT                            ;Stop the job

        END     RTEXEC
```

9.2 USING RTTRP AT THE INTERRUPT LEVEL

Your program can use the RTTRP call at interrupt level (that is, a realtime trap can contain an RTTRP monitor call) if the following rules are observed:

- The program must save any accumulators it may need; an interrupt-level trap overwrites them.
- The AC used in the interrupt-level RTTRP call cannot be accumulator 16 or 17.
- If the device given with the interrupt-level RTTRP is the same one that caused the interrupt, then no monitor calls of any kind can be executed in the routine. Execution of any further monitor calls (including RTTRP) dismisses the interrupt.

9.3 RELEASING REALTIME DEVICES

To release a realtime device, use the RTTRP monitor call with an argument list as follows:

```
addr:   EXP     Ø
        BLOCK   1
        EXP     <device>B9
```

Where device is the device name of the device to be released.

9.4 DISMISSING REALTIME INTERRUPTS

To dismiss a realtime interrupt, use the UJEN monitor call; this call causes the monitor to execute a JEN (or XJEN) instruction to the address saved by the previous JSR (to **trapaddr** or to the context switcher).

Executing any monitor call other than RTTRP during an interrupt dismisses the interrupt.

9.5 ASSIGNING RUN QUEUES

If your job has the JP.HPQ privilege, you can use the HPQ monitor call to place the job in a high-priority run queue. When your job executes a RESET or EXIT monitor call, the job returns to the queue given in the most recent SET HPQ monitor command.

As the monitor executes timesharing jobs, it scans high-priority queues before normal-priority queues; therefore jobs in high-priority queues are selected for execution before those in normal-priority queues.

Jobs in high-priority queues are also given preferential treatment by the monitor in its use of shared resources (such as shared device controllers). The monitor's swapper prefers high-priority jobs for first swap-in and for last swap-out.

The calling sequence for HPQ is:

```
MOVEI  ac,queue
HPQ    ac,
  error return
normal return
```

Where queue is the number of the high-priority queue that the job is to be placed in. If you give queue as Ø, the job returns to timesharing level. The highest queue number is a system configuration parameter (Ø-15).

9.6 SUSPENDING OTHER JOBS

If your job has the JP.TRP privilege, you can use the TRPSET monitor call to temporarily suspend execution of other jobs. This assures fast response times to realtime interrupts by minimizing contention for memory caused by other jobs' I/O.

The calling sequence for TRPSET is:

```
MOVE   ac,[XWD addr1,addr2]
TRPSET ac,
  error return
normal return
       . . .
```

where: addr1 is an address for storing an instruction (must be between 4Ø and 57 octal)
addr2 is the address of the argument list

The argument block addr2 points to is one of the following:

```
addr2:   JSR    addr3

addr2:   BLKI   addr3
```

where: addr3 is a user virtual address.

When the TRPSET call is executed, the monitor stops executing all other jobs. The contents of addr2 (a JSR or BLKI instruction) is moved to temporary storage, where the given address is converted to an executive virtual address. This address is then written into the location addrl.

The monitor also returns in the ac the previous contents of address; this allows your program to restore the contents of addrl after the TRPSET is executed.

The routine addressed during the interrupt must be locked into contiguous executive virtual memory; otherwise, the TRPSET call takes the error return.

On a multiprocessor system, the TRPSET call applies only to the processor specified as your job's CPU (use the SET CPU monitor command). The default is CPUØ.

The calling sequence for resuming other jobs is:

```
MOVEI  ac,Ø
TRPSET ac,
  error return   ;insufficient privileges
normal return    ;timesharing restarted
```

On a successful return, user I/O is set for the job.

CHAPTER 10

ANALYZING SYSTEM PERFORMANCE

The TOPS-10 monitor offers the following monitor calls to provide privileged users with tools to measure system performance: the PERF. call to control the system's performance meter and the SNOOP. monitor call to insert breakpoints in the monitor to trap to user programs.

10.1 THE PERFORMANCE FACILITY: PERF.

The PERF. monitor call allows privileged jobs on a KL10 processor to measure system performance over medium to long periods of time. See the Processor Reference Manual for a detailed presentation of the performance meter. PERF. allows you to count rates associated with memory cache, PI interrupt channels, program counters, microcode, hardware, and I/O channels

Only one job at a time can use the performance facility on any particular CPU. A job can have more than one CPU's PERF meter in use at the same time.

10.1.1 Performance Modes

The performance meter can operate in either of two modes:

- **Timer mode** (bit PM.MOD of .PMCPU is set) counts the number of CPU cycles while a condition specified by enabled bits is true. When the logical AND of all enabled bits is true, the counter counts at 1/2 the CPU cycle rate (model A=25MHz, model B=30MHz).

- **Counter mode** (bit PM.MOD of .PMCPU is cleared) counts the number of times the specified conditions occur.

10.1.2 Performance Enable Flags

The PERF. meter must be initialized before it can be started. When your job initializes the performance meter, it specifies which conditions are to be timed (timer mode) or counted (counter mode).

The meter has classes of conditions that you can enable by including flags in the argument list for .PRSET function of the PERF. UUO. The meter runs if all classes are true. A class is true if any condition within that class is true. For PERF., if no conditions within a class are specified, that class is ignored (forced true).

To obtain accurate measurement of the cache hit rate, set the PM.SYN bit in the cache enable word (.PMCSH); this synchronizes the performance and accounting meters.

Your job can measure the number of cache hits by using the performance meter to measure the total number of memory references and subtracting the number of cache misses. The cache hit rate is the number of cache hits divided by the total number of memory references.

To measure the cache hit rate for a job, the accounting meters must exclude both priority-interrupt time and monitor overhead. You can exclude these by rebuilding the monitor, after selecting the proper accounting options with MONGEN. Note, however, that these changes affect the methods for gathering usage accounting data.

Alternatively, you can set the following bits in GETTAB table .GTCNF:

Word	Symbol	Bit	Name	Meaning
17	%CNSTS	15	ST%EMO	Exclude monitor overhead from user runtime.
106	%CNST2	19	ST%XPI	Exclude priority interrupt time from user runtime.
106	%CNST2	20	ST%ERT	Use EBOX/MBOX accounting.

Your job must also set bit PM.NPI in the priority-interrupt enable word (.PMPIE); this enables the meter only when no interrupts are in progress.

To measure the cache hit rate for the system, the accounting meters must include both priority-interrupt time and monitor overhead. You can include these by rebuilding the monitor (selecting the proper accounting options), or by setting bit ST%XPI to 0 and ST%ERT to 1, in item %CNST2 of the GETTAB table .GTCNF. The performance meter must be set to measure cache misses, with no bits set in enable words 4 through 11 (.PMPIE through .PMCHN).

Note that the formats of accounting meter counts are different; to make them consistent, divide MBOX counts by 10000 (octal).

10.1.3 PERF. Functions

The following is a list of the PERF. functions:

Fcn-code	Symbol	Meaning
1	.PRSET	Sets up the performance meter.
2	.PRSTR	Starts the performance meter.
3	.PRRED	Reads the performance meter.
4	.PRSTP	Stops the performance meter.
5	.PRRES	Releases the performance meter.
6	.PRBPF	Turns background PERF analysis off.
7	.PRBPN	Turns background PERF analysis on.

The PERF. functions are usually used in the following order:

1. Initialize the performance meter (.PRSET).
2. Start the performance meter (.PRSTR).
3. Stop the performance meter (.PRSTP).
4. Release the performance meter (.PRRES).

Your program can also read the performance meter (.PRRED function) any time after initializing it, but before releasing it.

The calling sequence for the PERF. monitor call is:

```
          MOVE      ac,[XWD len,addr]
          PERF.     ac,
            error return
          normal return
          . . .
addr:     XWD       fcn-code,arglst
          . . .
          XWD       fcn-code,arglst
          . . .
arglst:   argument block
```

Where len is the length of the argument list and addr is the address of the argument list.

At the address you loaded into the ac, fcn-code is one of the PERF. functions listed above and arglst is the address of the argument list for the function code. This format allows you to specify multiple functions with a single PERF. call. At the address specified in arglst, store the argument block for the function code. Function codes and their argument blocks are described in the following sections.

10.1.3.1 Initializing the Performance Meter - Use the .PRSET function to initialize the performance meter. The format of the argument block, with offsets from the beginning of arglst (see above) for the .PRSET function is:

Word	Symbol	Contents
0	.PMLEN	Length of the argument block (not including this word).
1	.PMCPU	CPU type:

Bits	Symbol	Meaning
0	PM.PD6	PDP-6.
1	PM.KA	KA10.
2	PM.KI	KI10.
3	PM.KL	KL10.
4	PS.KS	KS10

Word	Symbol	Contents
2	.PMMOD	CPU number and mode:

Bits	Symbol	Meaning
Ø-17	PM.CPN	CPU number.
18	PM.MOD	Performance mode. If this bit is not set, the duration of time during which enabled events are true is counted. If this bit is set, the number of enabled events is counted. Refer to Section 1Ø.1.1.
19	PM.CLR	Clear performance meter counts.

3 .PMCSH Cache enable flags:

Bits	Symbol	Meaning
Ø	PM.CCR	Count references.
1	PM.CCF	Count fills.
2	PM.EWB	Count EBOX writebacks.
3	PM.SWB	Count sweep writebacks.
4	PM.SYN	Synchronize performance and accounting meters.

4 .PMPIE Priority interrupt enable flags:

Bits	Symbol	Meaning
Ø	PM.PIØ	Enable for channel Ø.
1	PM.PI1	Enable for channel 1.
2	PM.PI2	Enable for channel 2.
3	PM.PI3	Enable for channel 3.
4	PM.PI4	Enable for channel 4.
5	PM.PI5	Enable for channel 5.
6	PM.PI6	Enable for channel 6.
7	PM.PI7	Enable for channel 7.
8	PM.NPI	Enable for no interrupt in progress.

5 .PMPCE Program counter enable flags:

Bits	Symbol	Meaning
Ø	PM.UPC	User-mode enable.
1	PM.XPC	Executive-mode enable.

6 .PMMPE Microcode probe enable flag:

Bits	Symbol	Meaning
Ø	PM.MPE	Enable microcode probe.

7 .PMHPE Hardware probe enable flags:

Bits	Symbol	Meaning
Ø	PM.PØL	Probe zero low.
1	PM.PØH	Probe zero high.

Word	Symbol	Contents
10	.PMJOB	Job enable word. This contains the job number of the job for which the meter is enabled, or one of the following values:

Code	Symbol	Meaning
-2	.PMNUL	Enable for null job.
-1	.PMSLF	Enable for calling job.

Word	Symbol	Contents
11	.PMCHN	Channel enable flags:

Bits	Symbol	Meaning
0	PM.EC0	Enable for channel 0.
1	PM.EC1	Enable for channel 1.
2	PM.EC2	Enable for channel 2.
3	PM.EC3	Enable for channel 3.
4	PM.EC4	Enable for channel 4.
5	PM.EC5	Enable for channel 5.
6	PM.EC6	Enable for channel 6.
7	PM.EC7	Enable for channel 7.

10.1.3.2 Starting the Performance Meter - To start the performance meter, use the .PRSTR function. The argument block for this function is listed here with offsets from arglst (See Section 10.1.3.)

Word	Symbol	Contents
0	.PMLEN	Count of following words. (Supply this in your program.)
1	.PMCPN	CPU number. (Supply this in your program.)
2	.PMHTB	High-order word of time-base. (The rest of the data is returned by the monitor.)
3	.PMLTB	Low-order word of time-base.
4	.PMHPM	High-order word of performance counter.
5	.PMLPM	Low-order word of performance counter.
6	.PMHMC	High-order MBOX reference count.
7	.PMLMC	Low-order MBOX reference count.

Your program supplies the first two words of this argument block (.PMLEN and .PMCPN); the monitor returns the remaining words on a normal return.

10.1.3.3 Reading the Performance Meter - To read the performance meter, use the .PRRED function. The argument block for this function is the same as for the .PRSTR function. Your program supplies the first two words of the argument; the monitor returns the remaining words on a normal return.

10.1.3.4 Stopping the Performance Meter - To stop the performance meter, use the .PRSTP function. The argument block for this function is the same as for the .PRSTR function. Your program supplies the first two words of the argument; the monitor returns the remaining words on a normal return.

10.1.3.5 Releasing the Performance Meter - To release the performance meter, use the .PRRES function. The argument block for this function is the same as for the .PRSTR function. Your program supplies the first two words of the argument; the monitor returns the remaining words on a normal return.

10.1.4 Background PERF. Functions

When the performance meter is not assigned to a specific job, it can be set to gather system-wide statistics. This is called "background performance analysis." It is enabled using the PERF. function .PRBPF, and disabled with .PRBPN. While it is enabled, the monitor can sample PI channel and RH20 usage.

When setting the background performance meter, it is advisable to set bit 19 in .PMMOD to clear the counters. You set the timing interval in word .PSCSH, specifying the number of ticks.

The data for the background performance meter is kept in GETTAB table .GTCnV, where n is the CPU number. The table is formatted in 4-word blocks, of which the first pair of words contains the elapsed time since the time was cleared. The second pair of words in each block contains the meter count for the performance meter.

Because each CPU has one performance meter, the monitor samples each item in turn, for the specified period. At the end of the period, the meter updates the counter. The background performance meter is suspended if a job assigns the performance meter for another purpose. When it is deassigned, the background meter is restored.

10.1.5 PERF. Errors

On an error for a PERF. monitor call, one of the following error codes is returned in the ac:

Code	Symbol	Error
1	PRCPU%	Invalid CPU specified.
2	PRNXC%	Nonexistent CPU specified.
3	PRMOD%	Improper mode specified.
4	PRSET%	Meter not set up.
5	PRUSE%	Meter already in use.
6	PRRUN%	Meter already running.
7	PRJOB%	Invalid job number.
10	PRNRN%	Meter not running.
11	PRNIM%	Function not implemented.
12	PRFUN%	Invalid function code.
13	PRPRV%	Not enough privileges.

10.2 THE SNOOP FACILITY: SNOOP.

The SNOOP. UUO allows the privileged program ([1,2] or JACCT) to insert breakpoints into the monitor, thereby patching routines into the monitor code.

The breakpoints must be defined before they can be enabled. The SNOOP. function to define breakpoints requires your program to supply the checksum of the current monitor's symbol table. This requirement ensures that your program recognizes the current monitor symbols and locations. The checksum will be compared to the monitor's own analysis of the checksum, and if the two are not equal, the SNOOP. UUO will fail.

In addition, the breakpoint definitions must supply monitor addresses for locations that will be patched. These addresses can be obtained from the monitor symbol table as well.

To obtain the checksum and monitor addresses, your program can use its own algorithms, or you can call routines from the SNUP.MAC package that is distributed with the monitor sources. SNUP.MAC contains several routines that obtain information to be used in the SNOOP. argument block, and others serve as examples of how to use the SNOOP. UUO.

To compute the checksum and locations yourself, you must obtain the file specification of the current monitor. The relevant information can be obtained from GETTAB table .GTCNF, where the important words are:

Word	Contents
%CNBCP	Bootstrap CPU number.
%CNBCL	Bootstrap line number.
%CNNCR	Number of CPUs allowed to run.
%CNMBS	File structure where bootstrap monitor is stored.
%CNMBF	File name of bootstrap monitor.
%CNMBX	File extension of bootstrap monitor.
%CNMBD	Bootstrap file directory.

The maximum number of breakpoints that can be inserted is defined as GETTAB symbol %CNBPM in table .GTCNF; the default monitor defines this value to be 64.

To insert the defined breakpoints, your program must be locked into contiguous Executive Virtual Memory (EVM). You can lock your job into EVM using the LOCK. UUO, which will return the new starting address (as the virtual page number) of your job. Remember that subsequent references to user address space must be accompanied by the offset of the relocation. That is, you should compute the difference between your original starting address in user space and the new starting address in EVM, and use that difference as an offset for references to locations in your program.

Because you must lock your job in EVM to use the SNOOP. UUO, you must remember that the monitor will perform no functions to save data when you pass control to the routines in your program that perform the actually data gathering and analysis. In other words, save the state of the current accumulators when you JRST to your routine at the breakpoint. Observe that the monitor defines accumulator P=1, not 17, as user programs normally do. While in breakpoint code, your program cannot execute a monitor call or leave exec mode.

After inserting the breakpoints, you can remove them at will with SNOOP. UUO. After you remove them, you can re-enable them, or you can delete the breakpoint definition. To delete a breakpoint definition, you must remove the breakpoint first. Similarly, to insert a breakpoint, you must define it first.

As with real-time job execution, only one job can define breakpoints at a time. Until the job undefines its breakpoints, all other jobs attempting to snoop will receive an error. In the event of necessary recovery functions, the monitor can delete the breakpoint definitions. This might occur if a memory parity error occurred, or if your job received a stopcode. When your program issues a RESET, that also removes and deletes definitions of breakpoints.

Examples of using SNOOP. are provided in the description of the monitor call in Chapter 22.

The SNOOP. functions are listed here. More specific explanation of their action is given in following sections.

Fcn-code	Symbol	Meaning
Ø	.SODBP	Define breakpoints.
1	.SOIBP	Insert breakpoints.
2	.SORBP	Remove breakpoints.
3	.SOUBP	Delete breakpoint definitions.
4	.SONUL	Null function. This function is useful when you must patch code into the monitor to perform functions at UUO level that cannot be accomplished at interrupt level.

The calling sequence for the SNOOP. call depends on the function code. Only the function to define breakpoints (.SODBP) requires an argument block. After the breakpoints have been defined, the subsequent functions of the SNOOP. call will take action on the breakpoints you have defined.

Each function and calling sequence is described in the following sections.

10.2.1 Defining Breakpoints

The SNOOP. function Ø allows you to define the breakpoints that you will insert into the monitor code. The calling sequence for .SODBP is:

```
MOVE      ac,[XWD .SODBP,addr]
SNOOP.    ac,
  error return
normal return
```

In this calling sequence, the left half of the ac is loaded with the function code (in this case, .SODBP, function code Ø). The right half is loaded with addr, the address of the argument list. On an error, the SNOOP. call takes a non-skip return, and the error code is stored in the ac. (Error codes are listed in Section 10.2.5.) No breakpoints will be defined on an error. If, however, the call takes a successful (skip) return, all the breakpoints in the argument list will be defined. You can issue the SNOOP. call with function code 1 to insert the breakpoints, enabling the breakpoint facility.

The argument list for SNOOP. function Ø (.SODBP) is:

Word	Symbol	Contents
Ø	.SOLEN	The total length of the argument list, including this word.
1	.SOMSC	The checksum of the monitor symbol table.

Following these is a two-word pair for each breakpoint you want to define. Therefore, the total number of breakpoints will be the result of: {[.SOLEN minus 2] divided by 2}.

Word	Symbol	Contents
2	.SOMVA	The monitor virtual address where the breakpoint instruction is to be inserted. This address must be obtained from the monitor symbol table to ensure accuracy.
3	.SOBPI	The breakpoint instruction that is to be inserted at the monitor address specified in .SOMVA.

Repeat words .SOMVA and .SOBPI for each breakpoint to be defined.

Note that the inserted instruction is executed prior to the original instruction. If the inserted instruction skips on return, the original instruction will not be executed. If the inserted instruction skips more than one instruction, or does not return, the SNOOP. data base will be damaged.

1Ø.2.2 Inserting Breakpoints

After your program defines the breakpoints, they can be inserted with SNOOP. function 1, .SOIBP. Your program must be locked in contiguous EVM to use this function. (Refer to the LOCK. UUO.) The calling sequence for the .SOIBP function is:

```
MOVE      ac,[XWD .SOIBP,Ø]
SNOOP.    ac,
  error return
normal return
```

The .SOIBP function does not require an argument list. If the error return is taken, the error code is returned in the ac. If the normal return is taken, the breakpoints were inserted and are enabled.

1Ø.2.3 Removing Breakpoints

Breakpoints that have been inserted with the .SOIBP function can be removed with the .SORBP function (function code 2). This function will succeed if the breakpoints have been inserted, and will remove all the breakpoints. The calling sequence for this function is:

```
MOVE      ac,[XWD .SORBP,Ø]
SNOOP.    ac,
  error return
normal return
```

After removing the breakpoints, you can insert them again (using function .SOIBP), or you can delete the breakpoint definitions (using function .SOUBP).

10.2.4 Deleting Breakpoint Definitions

After breakpoints have been removed (using function .SORBP), the breakpoint definitions can be deleted. This clears the definitions that you made with the .SODBP function. The calling sequence for this function is:

```
MOVE      ac,[XWD .SOUBP,0]
SNOOP.    ac,
  error return
normal return
```

This function will delete all definition of the breakpoints, and you must redefine them by using .SODBP if you want to insert them again. Note that breakpoint definitions can be deleted by the monitor and by a RESET from your program (as described in Section 10.2).

10.2.5 SNOOP. Error Codes

The following is a list of the error codes that can be returned in the ac after a SNOOP. UUO fails:

Code	Symbol	Error
1	SOIAL%	Illegal argument list.
2	SONPV%	Not enough privileges.
3	SOSAS%	Another program is already SNOOPing.
4	SOMBX%	Maximum number of breakpoints is exceeded.
5	SOIBI%	Breakpoints are already inserted.
6	SONFS%	No monitor free core is available.
7	SOADC%	Address check.
10	SOINL%	Program is not locked in contiguous EVM.
11	SOWMS%	Monitor symbol table checksum does not match.

CHAPTER 11

PROGRAM INPUT AND OUTPUT

This chapter describes I/O programming in general terms. Further discussions of I/O programming for specific devices are included in the chapters describing devices.

11.1 OVERVIEW OF THE I/O PROCESS

To perform I/O, your program should:

- Initialize the program.
- Initialize a device.
- Initialize a buffer ring when using buffered I/O or define a command list when using dump I/O.
- Select a file.
- Transmit/receive data.
- Close the file.
- Release the device.
- Stop the program.

There are two ways of performing I/O: buffered I/O and dump I/O. The list below summarizes the monitor calls you need to perform I/O with each method. Each of the monitor calls listed below in the summary is described in detail in Volume 2, Chapter 22. Note that any device operation listed below can be done with the FILOP. call, because I/O operations using channels 20-117 can only be done with FILOP.

Step	Buffered I/O	Dump I/O
Program Initialization	RESET	RESET
Device Initialization	INIT, OPEN, FILOP.	INIT, OPEN, FILOP.
Buffer Initialization	INBUF, OUTBUF, FILOP.	
File Selection	LOOKUP, ENTER FILOP.	LOOKUP, ENTER FILOP.
Device Position	USETI, USETO UGETF, SUSET., FILOP., MTAPE	USETI, USETO UGETF, SUSET., FILOP., MTAPE
Data Transmission	IN, INPUT, FILOP. OUT, OUTPUT	IN, INPUT, FILOP., OUT, OUTPUT
Command List Definition		command list
File Termination	CLOSE, FILOP.	CLOSE, FILOP.
Device Termination	RELEASE, FILOP.	RELEASE, FILOP.
Program Termination	EXIT	EXIT

In the summary above, when a group of monitor calls is listed for a given I/O step, usually only one of the calls from the group must be included in your program. For example, the data transmission step lists four calls (IN, INPUT, OUT, and OUTPUT). However, only one of these is present in your program for each data transmission. On the other hand, there are three calls listed for the file selection step. At times you should include more than one of these calls in your program for proper file selection.

The following sections in this chapter describe the calls that are necessary for each step. Note that the file selection step and the device position step apply only to directory devices (DECtape, labelled magtape, and disk). Unlabelled magnetic tapes can be positioned using the MTAPE or TAPOP. monitor calls. The device positioning monitor calls, however, may be included in your program for any device, allowing device interchangeability, because the monitor ignores calls which are inappropriate for the device.

11.2 INITIALIZING A PROGRAM

To initialize a program, use the RESET monitor call. This call performs most of the functions you will want when initializing your program. The RESET call should be the first monitor call in your program.

11.3 INITIALIZING A DEVICE

To initialize a device, use the OPEN, INIT, or FILOP. monitor call. Each of these calls can initialize a channel for the device, making it available for use in your program. Your program can use up to 16 I/O channels (Ø-17 octal) using OPEN or INIT. The extended channel facility, which you use with FILOP. allows you to used up to 64 (decimal) additional I/O channels (providing a total of up to 8Ø channels).

11.3.1 TOPS-10 Devices

The TOPS-10 operating system supports the following devices. Note, however, that current versions of TOPS-10 restrict the support status of some devices. For information about the current support status of TOPS-10 devices, refer to the TOPS-10 Software Product Description.

- Disks (DSK). Refer to Chapter 12.
- DECtapes (DTA). Refer to Chapter 13.
- Magtape units (MTA). Refer to Chapter 14.
- Terminals (TTY). Refer to Chapter 15. This chapter also discusses the pseudo-terminal (PTY) device. The remote data terminal (RDx) device is discussed in Chapter 21.
- Line printers (LPT). Refer to Chapter 16.
- Card readers (CDR) and card punches (CDP). Refer to Chapter 17.
- Papertape readers (PTR) and papertape punches (PTP). Refer to Chapter 18.
- Plotters (PLT). Refer to Chapter 19.
- Display light pens (DIS). Refer to Chapter 20.
- Network nodes, using the TSK logical device. Refer to Chapter 5.
- Multiplexed devices (MPX). Refer to Section 11.3.4.

This chapter contains information common to all these devices.

When the system is loaded, most devices are assigned to the monitor's pool of available resources. When a program requests a device, the monitor assigns it to the program; when the program releases the device, the monitor returns it to the pool.

Most devices can be used interchangeably during program execution. Therefore you can write a program for one device, and substitute a different device during program execution. This substitution can be accomplished by the ASSIGN command, or by the REASSI or DEVLNM monitor call.

Specifically, TOPS-10 devices may be divided into "directory devices" and "non-directory devices." A directory device contains named files stored in directories. Disk, DECtape, and labelled magtapes are directory devices. A DECtape contains one directory; disk file structures and magtapes may contain multiple directories. Therefore, to access data on disk, a file name and directory specification might be required. TOPS-10 allows you to program I/O regardless of the type of device. You should include the file name and directory for all references to files. For non-directory devices, the monitor ignores directory information.

11.3.2 Device Names

Your program refers to devices by their device names. The various formats of device names are described below. In this description, dd and ddd are 2- and 3-letter mnemonics, nn is a 2-digit node number, and u is a 1-digit unit number. Note that many system and user programs require a trailing colon for parsing; however, monitor calls do not allow the trailing colon in device names.

Format	Example	Meaning
d		Do not use this format.
dd	LL	The monitor tries to select a device of the type specified (in the example, a lowercase line printer) at the job's node. If all such devices are in use, the monitor takes the error return for the monitor call; if no such device exists at the job's node, the monitor tries to select a device at the central site. If the job has such a device assigned but not initialized, the monitor selects that device.
ddnn	LL23	The monitor tries to select a device of the type specified at the specified node (in the example, a lowercase line printer at node 23).
ddnnu		Do not use this format.
ddu		Do not use this format.
ddd	LPT	The monitor tries to select a device of the type specified at the job's node (in the example, a line printer). See the meaning of dd above. If all such devices are in use, the monitor takes the error return for the monitor call; if no such device exists at the job's node, the monitor tries to select a device at the central site.
dddnn	LPT23	The monitor tries to select a device of the type specified at the specified node (in the example, a line printer at node 23). See the meaning of ddnn above.
dddnnu	LPT231	The monitor tries to select the specified device at the specified node (in the example, line printer number 1 at node 23).

Format	Example	Meaning
dddu	LPT3	The monitor tries to select the specified device at the job's node (in the example, line printer number 3 at the job's node). If all such devices are in use, the monitor takes the error return for the monitor call; if no such device exists at the job's node, the monitor tries to select a device at the central site.
ddnuuu	RAD222	The monitor tries to select the dd disk device, unit uuu, on the HSC-50 node n. In this case, the device is an RAxx type, the unit is 222, and the HSC-50 node is 4 (where A=1, B=2, and so on). Unit numbers for disk are always decimal.

There are four types of device names:

- Generic names specify general types of devices. When you specify a generic device name, the monitor chooses the first available unit that satisfies the requirements of the generic device name.
- Physical names specify device units, possibly on specific controllers, allowing you to specify the exact unit your job requires.
- Logical names are user-defined or monitor-defined names for devices. Monitor-defined logical names are specific devices that have a special purpose in the system.
- Ersatz names are logical device names assigned by the monitor for disk areas that have a particular purpose, such as libraries. This allows you to specify a disk area with a short device name.

11.3.2.1 Generic Device Names - A generic device name is the most general name of a device. When your program specifies a generic device name, the monitor tries to select a free device of the specified type at the job's node; if none is available, the monitor tries to select a free device of the type specified at the central site.

A generic device name is a 2- or 3-letter string. Note that many system and user programs require a trailing colon for parsing; however, monitor calls do not allow the trailing colon in device names.

The generic device names and their meanings are:

3-Letter Name	2-Letter Name	Meaning
TTY	TT	User terminal.
DSK	DS	Disk. The monitor uses the job search list to select the device. See Chapter 13 for a discussion of job search lists.
MTx	MT	Magtape unit on controller **x**. For example, MTA specifies a magtape unit on controller A.
	M7	7-track magtape.
	M9	9-track magtape.
DTx	DT	DECtape unit on controller **x**. For example, DTA specifies a DECtape unit on controller A.
LPT	LP	Line printer.
	LL	Uppercase and lowercase line printer.
	LU	Uppercase line printer.
CDR	CR	Card reader.
CDP	CP	Card punch.
PTR	PR	Papertape reader.
PTP	PP	Papertape punch.
PLT	-	Plotter.
DIS	-	Display.
PTY	-	Pseudo-terminal.
RDx	-	Remote data terminal, where **x** is the controller name (A to Z).

11.3.2.2 **Physical Device Names** - Every device has a physical device name of the form:

dddnnu

where **ddd** is a 3-letter generic device name, **nn** is a node number, and **u** is a unit number. For example, LPT231 specifies line printer number 1 at node 23. This convention does not apply to TTYs, however.

11.3.2.3 **Logical Device Names** - A logical device name is an alphanumeric string of up to six characters, assigned by the monitor or by the user. The monitor has the following predefined logical device names to refer to specific devices that are used for the following purposes:

- OPR specifies the physical terminal designated by the operator as the operator terminal.
- NUL specifies a null device. Output to NUL is lost. If you attempt input from NUL; your IN UUO fails and GETSTS returns IO.EOF (end-of-file) in the file status word.

- CTY specifies the console terminal for the system.
- TTY specifies your job's terminal.

User-defined logical names are defined by the ASSIGN or MOUNT monitor command, or by the REASSI or DEVLNM monitor call. You can assign one logical name to each nondisk physical device.

A logical device name takes precedence over a physical device name. Therefore, if your program defines the logical name DSK for a magtape device, all references to DSK specify the magtape rather than the user disk area.

There are methods for indicating that the monitor should ignore logical name definitions. This can be specified in the OPEN call, by setting the bit UU.PHS in the argument block. (Note that this bit can also be set in FILOP. word .FOIOS).

Alternatively, you can disable logical name definitions for a particular CALLI function by XORing Bit 19 (UU.PHY) into the CALLI monitor call. For example, to obtain the device type of a physical device LPT, use the following sequence:

```
MOVE T1,[SIXBIT/LPT/]
DEVTYP T1,UU.PHY
```

This calling sequence ORs UU.PHY into the argument of the DEVTYP call, with the device name LPT. The information will be returned for physical device LPT.

For negative CALLIs, you must place the negative value of UU.PHY in the address field. For the LIGHTS monitor call (CALLI -1), and customer-defined monitor calls (CALLI -2 and less), use the following sequence to perform the call, ignoring logical names:

```
LIGHTS T1,-UU.PHY
```

11.3.2.4 Ersatz Device Names - An ersatz device is a disk-simulated library. Although an ersatz device is OPENed like a directory device, an ersatz device represents a particular project-programmer number on disk structures that are specified in a search list. The particular search list used depends on the ersatz device specified in your program. The following is a list of TOPS-10 standard ersatz device names. It shows the ersatz device name, the PPN to which the name is assigned (if any), the search list of file structures on which the PPN exists, and the intended use of the disk area.

Table 11-1: Ersatz Devices

Name	PPN	Search List	Purpose of Area
ACT	1,7	SSL	System accounting
ALG	5,4	SSL	ALGOL
ALL	NONE	ALL	All structures
APL	5,31	SSL	APL
BAS	5,1	SSL	BASIC
BLI	5,5	SSL	BLISS
COB	5,2	SSL	COBOL
CTL	5,27	SSL	Control files
D60	5,32	SSL	DN60 (IBMCOM) files
DBS	5,24	SSL	DBMS
DEC	10,7	SSL	Field-image versions of programs
DMP	5,21	SSL	Dump area
DOC	5,14	SSL	Documentation
DSK	NONE	JSL	Default device
FAI	5,15	SSL	FAIL
FFA	1,2	SSL	Operator (Full File Access)
FOR	5,6	SSL	FORTRAN
FNT	5,36	SSL	LN01 fonts
GAM	5,30	SSL	Games
HLP	2,5	SSL	HELP files
INI	5,34	SSL	Initialization files
LIB	SETTABLE	JSL	Library
MAC	5,7	SSL	MACRO
MFD	1,1	SSL	Master file directory
MIC	5,25	SSL	MIC files
MUS	5,16	SSL	Music
MXI	5,3	SSL	PDP-11
NEL	5,20	SSL	NELIAC
NEW	1,5	SSL	New CUSPs
OLD	1,3	SSL	Old CUSPs
POP	5,22	SSL	POP2
PUB	1,6	SSL	User maintained programs
REL	5,11	SSL	.REL files
RNO	5,12	SSL	RUNOFF files
SNO	5,13	SSL	SNOBOL
SPL	3,3	SSL	Spooling area
SSL	NONE	SSL	System search list
STD	1,4	SSL	Same as SYS, but /NEW doesn't apply
SYS	1,4	SSL	System CUSPs
TED	5,10	SSL	Text editor
TPS	5,26	SSL	Text processing
TST	5,23	SSL	Test area
UMD	6,10	SSL	User mode diagnostics
UNV	5,17	SSL	Universal files
UPS	5,35	SSL	Mail system area
UTP	5,33	SSL	UETP
XPN	10,1	SSL	Crash dumps

The search lists in this list are:

- **SSL** (System Search List) is defined by the system manager while running the ONCE-only startup dialog, using the LONG startup option. Refer to the TOPS-10 Software Installation Guide for more information.
- **JSL** (Job Search List) is defined as equivalent to the SSL by default; the default can be changed using the SETSRC program. Refer to the TOPS-10 User Utilities Manual for more information on SETSRC.
- **ALL** (All Search List) is the definition of all the public file structures in their order of accessibility.

11.3.2.5 Pathological Device Names - The user can define a logical name for a directory search path. Such user-defined names are called "pathological names." Pathological names can define one or more disk directories, and are described in Section 12.6.5.

11.3.3 Universal Device Indexes

The monitor maintains an index to all active devices; this index is called the Universal Device Index (UDX). For Versions 7.02 of TOPS-10 and later, the description of the Universal Device Index is fixed only for terminal devices. Many operations allow your program to use the Universal Device Index for a device rather than its name or channel number. Use the IONDX. monitor call to obtain the Universal Device Index for a device.

The format of a Universal Device Index is:

Bits	Symbol	Meaning
18-20		Device or process:

Code	Symbol	Meaning
0	.UXCHN	Physical device.
2	.UXTRM	Terminal.
3	.UXPRC	Process.

Bits	Symbol	Meaning
21-26	UX.TYP	Device type. Refer to the DEVTYP call in Volume 2, Chapter 22, for the list of device types. This field is 0 for termials.
27-35	UX.UNT	Unit number within type.

For example, the Universal Device Index 200114 indicates the device TTY114; the 2 in Bits 18-26 shows that the device is a terminal, and the 114 in Bits 27-35 shows that it is TTY114.

11.3.4 MPX-Controlled Devices

An MPX (multiplexed) channel can have many devices connected to it. (Refer to the CNECT. monitor call.) A device connected to an MPX channel is an MPX-controlled device. Only terminals, line printers, papertape punches, card readers, remote data terminal (RDx), and pseudo-terminals are MPX-controllable.

Your programs can refer to MPX-controlled devices by their physical or logical names, or by their Universal Device Indexes.

11.3.5 Spooled Devices

Some devices can be spooled by the monitor. This means that input from or output to these devices is not necessarily done immediately. Instead, the data is stored in a special system area and is input or output at a convenient time.

Devices that can be spooled are: card reader, line printer, card punch, papertape punch, and plotter. The spoolers may be controlled by GALAXY, which is documented in the TOPS-10 Operator's Guide. For the spooled devices for your job, the GETTAB table .GTSPL contains a bit for each type of device. If the bit is on and the device is ASSIGNed or INITed for your job, the device is spooled. When you CLOSE or RELEAS a spooled output device, output is saved until the device becomes available. The I/O is copied to disk, to the DSK:[3,3] (spooling) area.

For example, if the system card reader is spooled, the monitor reads any available input from the card reader into its storage area. When a program requests card reader input, the monitor supplies the data previously read.

For another example, if the system line printer is spooled, output for the line printer is placed in the monitor's storage area. Then the system prints the data at a convenient time.

You spool the device using the SET SPOOL monitor command or the SETUUO monitor call. Frequently, LOGIN spools devices under control of the accounting files.

For input spooling, if a LOOKUP is given after the cardreader is initiated with the INIT call, the LOOKUP is ignored and an automatic LOOKUP is done, using the file name given in the previous SET CDR monitor command with the file extension of .CDR. After every automatic LOOKUP, the name in the input-name counter .GTSPL is incremented by 1, so that the next automatic LOOKUP will use the correct file name.

For output spooling, if an ENTER is done, the file name is stored in the RIB in location .RBSPL so that the output spooler can label the output. Therefore, programs should specify a file name if possible. If an ENTER is not done, an automatic ENTER is generated, using a file name in the form:

```
xxxyyy.zzz
```

Where xxx is a unique, three-character name devised by the monitor, yyy is either of the following:

- Snn, where nn is the ANF-10 remote station node number for a generic device.
- The unit number of the specified unit.

and zzz is a three letter generic device name, such as LPT. Under GALAXY 4.1 and later versions, the file names for spooled files are generated as:

filename.SPL

For previous versions of GALAXY or MPB, the file names are:

filename.CDP for card punch
filename.CDR for card readers
filename.LPT for line printers
filename.PLT for plotters
filename.PTP for papertape punch

11.3.6 Restricted Access Devices

Any device except a disk or a terminal can be restricted to controlled access. This means that the operator (or a privileged user) can make the device assignable only by the operator. Privileged users can use the DVRST. monitor call to designate a specified device as being restricted and the DVURS. call to remove the restricted status of a device.

The GALAXY 4.1 batch and spooling system controls devices with the Mountable Device Allocator (MDA) routines. Control by MDA is set with the DEVOP. function .DVMDS, and cleared using function .DVMDC. This MDA-controlled bit is set to prevent assignment of devices (except disk devices) by user jobs. MDA control prevents access by programs using the DVURS. call.

To request a restricted device, give the MOUNT monitor command. The operator or GALAXY system can then send you a message granting or refusing your request.

To release a restricted device, give a DISMOUNT, DEASSIGN, or FINISH command, or log off the system. This returns all devices assigned to your job to the monitor.

You can request unrestricted devices for use either by your job or by a program. To request a device for your job, give the ASSIGN monitor command or use the REASSI UUO. You will receive a message confirming or denying your request. To request a device for use by a program, use the OPEN, INIT, or FILOP. monitor call.

To release an unrestricted device from your job, give the DEASSIGN or FINISH command, or log off the system. To release an unrestricted device from your program, use the RELEAS, RESET, or EXIT monitor call.

11.4 MODES

You can use eleven data modes when performing I/O. These data modes are:

*ASCII
*8-bit ASCII
*ASCII Line
*Packed Image
*Byte
*Image
*Image Binary
*Binary
Image Dump
Dump Records
Dump

Data modes preceded by an asterisk (*) transmit data in buffered I/O mode. Packed Image mode transmits data in buffered I/O mode for terminals only. The remaining data modes do not use the normal buffering scheme. These data modes are sometimes referred to as unbuffered modes or, more commonly, dump I/O data modes.

All data transmissions are performed in either buffered I/O mode or dump I/O mode. Your program specifies the mode of transmission in an argument to the INIT, OPEN, FILOP., or SETSTS monitor call. The data modes are listed in Table 11-2, and every data mode that is applicable to each device is described in the chapter describing the device.

Table 11-2: Data Modes

(Bits 32-35 of the file status word)

Code	Symbol	Data Mode	Applicable Devices
Ø	.IOASC	ASCII	Disk Terminal Magnetic Tape DECtape Plotter Card Punch Card Reader Line Printer Paper Tape Punch Paper Tape Reader
1	.IOASL	ASCII Line	Disk Terminal Magnetic Tape DECtape Plotter Card Punch Card Reader Line Printer Paper Tape Punch Paper Tape Reader
2	.IOPIM	Packed Image	Terminal
3	.IOBYT	Byte	Magnetic Tape ANF-1Ø Network Task
4	.IOAS8	8-bit ASCII	Terminal Network Line Printer PTY
5		Reserved for DEC	
6-7		Reserved for Customer	

Table 11-2: Data Modes (Cont.)

(Bits 32-35 of the file status word)

Code	Symbol	Data Mode	Applicable Devices	
10	.IOIMG	Image	Disk Terminal Magnetic Tape DECtape Plotter	Card Punch Card Reader Line Printer Paper Tape Punch Paper Tape Reader
11-12		Reserved for DEC		
13	.IOIBN	Image Binary	Disk Magnetic Tape Line Printer DECtape Plotter	Card Punch Card Reader Paper Tape Punch Paper Tape Reader
14	.IOBIN	Binary	Disk Magnetic Tape Line Printer DECtape Plotter	Card Punch Card Reader Paper Tape Punch Paper Tape Reader
15	.IOIDP	Image Dump Mode	Display	
16	.IODPR	Dump Record	Disk Magnetic Tape	DECtape
17	.IODMP	Dump	Disk Magnetic Tape	DECtape

11.5 DEFINING A COMMAND LIST

Image Dump, Dump Record, and Dump modes are nonbuffered I/O modes. These modes use a command list that specifies the area in your allocated memory to be read or written. The address of the command list is specified in your program with either an IN or INPUT monitor call for input or an OUT or OUTPUT monitor call for output.

The command list must exist in your program's low segment. Take care not to load the command list into the high segment. Otherwise, your program will be stopped when the IN, INPUT, OUT, or OUTPUT call is executed and the following message will be printed by the monitor:

?Address check for device device; UUO at user PC addr

Your program specifies the first word of the command list in a FILOP., IN, INPUT, OUT, or OUTPUT monitor call. Only three types of entries can appear in a command list. These entries are:

- An I/O instruction, in IOWD format
- A GOTO instruction
- A terminator, in the form of a zero word

An IOWD instruction causes a specified number of words to be transmitted to or from a specified location in your program. Its format is:

```
IOWD n,loc
```

The n specifies the number of words to be transmitted, and loc specifies the location of the first word to be transmitted. After the specified data has been transmitted, the monitor finds the next command at the location immediately following the IOWD in your program. Note that the following are equivalent:

```
IOWD n,loc = XWD -n,loc-1
```

Note that every IOWD that transmits data to the disk writes a separate record on the disk.

The GOTO command specifies the next command to be executed. Its format is:

```
XWD Ø,y
```

The y specifies the location of the next command. A maximum of three consecutive GOTOs are permitted in a command list. After the third consecutive GOTO, your program must include an IOWD that transfers data.

The zero word must be present in the command list to terminate the list. When the command list has been completely processed, the monitor returns control to your program. However, if the monitor encounters an illegal address while processing the command list, the monitor stops your job and prints the following message on your terminal:

```
?Illegal address in UUO at user PC addr
```

Below are some sample command lists that can be used for dumping I/O to disk:

```
OUTPUT DISK,[IOWD 7Ø,BUF1     ;Transmit 7Ø words beginning at BUF1
             IOWD 7Ø,BUF2     ;Transmit 7Ø words beginning at BUF2
             Z]               ;A zero word
```

Another example is:

```
        OUTPUT DSK,ADDR1      ;ADDR1 is address of command list
        .
        .
        .
ADDR1:  IOWD 7Ø,BUF1          ;Transmit 7Ø words beginning at BUF1
        IOWD 7Ø,BUF2          ;Transmit 7Ø words beginning at BUF2
        Ø                     ;A zero word
```

Below is an example of dump mode input:

```
        TITLE   HOME - Routine to read HOME block
        SUBTTL  HANLEY A. STRAPPMAN 6-27-83/HAS
        SEARCH  UUOSYM

;AC'S
        T1=1                            ;Temp
        T2=T1+1
        P=17                            ;Stack pointer

;MISCL
        HOMNAM==0                       ;SIXBIT/HOM/
        HOMCOD==176                     ;Code number
        CODHOM==707070                  ;Code for HOME
        BLKSIZ==200                     ;Size of a disk block
        II==0                           ;Channel
;Routine to read the HOME block
;T1 passes the structure name
;The HOME block is returned in BF
;Skip if successful
HOME::  MOVEM   T1,DEV+.OPDEV           ;Store str name
        OPEN    II,DEV                  ;Open str
          POPJ  P,
        LOOKUP  II,FIL                  ;LOOKUP HOME.SYS
          POPJ  P,
LOOP:   IN      II,CMD                  ;Read a block
          SKIPA T1,BF+HOMCOD            ;Pick up code word
        POPJ    P,                      ;EOF
        MOVS    T2,BF+HOMNAM            ;Pick up name
        CAIN    T1,CODHOM               ;Right block?
        CAIE    T2,'HOM'
        JRST    LOOP                    ;No, keep searching
        RELEAS  II,                     ;Release channel
        AOS     (P)                     ;Skip return
        POPJ    P,

DEV:    .IODMP                          ;OPEN block
        BLOCK   1
        0
FIL:    .RBEXT                          ;LOOKUP block
        XWD     1,4
        SIXBIT  /HOME/
        SIXBIT  /SYS/
CMD:    IOWD    BLKSIZ,BF               ;I/O command list
        0
BF::    BLOCK   BLKSIZ                  ;Buffer
        END
```

11.6 SELECTING A FILE

There are several monitor calls you can use to select a file for use in your program:

- To create or write a new file, use the FILOP. or ENTER monitor call. This enters the file name into the directory. You can then (optionally) use a USETO monitor call to specify a block number for file output. Then use OUT or OUTPUT monitor calls to write the file.

- To delete an existing file, use the LOOKUP monitor call to identify the file; then use the RENAME monitor call with a zero file name to delete the file. If the program must read the file before it is deleted, use the IN or INPUT monitor calls before the RENAME call.

- To read a file, use the LOOKUP monitor call to identify the file. You can then (optionally) use the USETI call to specify a block number within the file. Read from the file using the IN or INPUT monitor calls.

- To update a file, use the LOOKUP and ENTER monitor calls to identify the file. You can then (optionally) use the USETI and USETO calls to specify block numbers within the file. Use the IN or INPUT calls to read from the file, and the OUT or OUTPUT calls to write into the file.

- To change the attributes of a file, use the LOOKUP and RENAME monitor calls, giving the desired new attributes.

- To rename a file, use the LOOKUP monitor call to identify the file; then use the RENAME call to define its new name.

- To append to a file, use the LOOKUP and ENTER monitor calls on the same I/O channel to identify the file. Then skip over the old part of the file by using a USETO call to the end of the file. The FILOP. monitor call may also be used; it will do all the skipping for you. Append new data to the file by using the OUT or OUTPUT calls.

- To supersede a file, use the ENTER monitor call to identify the file. Then write the new file by using OUT or OUTPUT calls.

11.7 TRANSMITTING DATA

To transmit data to and from files, use the IN, INPUT, OUT, and OUTPUT monitor calls. These calls can be used only for initialized channels.

You can select a block number within a file for input or output by using the USETI and USETO monitor calls (disk and DECtape devices only).

11.7.1 Output (Writing a File)

The monitor controls I/O in response to monitor calls issued from your program through which data may be passed between memory and disk. As many as 80 (decimal) channels are available to each user. Note that Channels 20-117 (octal) can only be used with the FILOP. call.

The request to open a channel is made with the OPEN call. You would proceed by issuing:

```
OPEN channo,OPNBLK
```

Where **channo** is the I/O channel number and **OPNBLK** is the starting address of an argument list. At the location starting with OPNBLK, you must reserve three words.

In the first word, among other things, you must declare the mode of data transmission. Data transmission is always in bytes and the mode designates the byte size that is to be used. Suppose, for example, that the first word of OPNBLK contains Ø. Data transmission is then in 7-bit ASCII mode (IO.ASC), five characters to a word. All available data modes are listed in Section 11.4.

The second word, OPNBLK+1, contains the device name. If you wish to do disk I/O, the name for the disk is DSK, entered in SIXBIT.

The third word contains the addresses of the ring headers (used only in buffered mode). The format of the buffer ring header is described in Section 11.9. The left half specifies the address of the output ring header. The right half specifies the address of the input ring header. For present purposes, we shall not perform input, so we set the right half of OPNBLK+2 to zero. Thus, our 3-word block looks as follows:

```
OPNBLK: Ø
        SIXBIT   /DSK/
        XWD      OUTB,Ø
```

The next monitor call to be issued in connection with your output is the OUTBUF call. The monitor responds to this call by reserving a block of memory locations. The monitor transmits data in blocks, with the size of each block depending on the destination device. A block of data on the disk contains 2ØØ octal words. An area of memory is used by the monitor to hold data until a full block is collected for output to the disk. Data is written to disk in multiples of 2ØØ-word blocks. When a program writes a partial block, the remainder of the block is filled with zeros.

For directory devices, you must now specify the name of a file to which your data is to be output. For this, use the ENTER call, which causes the monitor to store a directory entry for subsequent use. The ENTER call is formatted as:

```
ENTER    channo,entblock
```

where **entblock** is the starting address of an argument list that you set up for reference by the monitor when it executes the ENTER call.

The argument block for the ENTER UUO is called the LOOKUP/ENTER/RENAME block because the same type of block is used for output operations and input operations. The LOOKUP/ENTER/RENAME argument block can be given in either of two forms: a four-word argument block (also called the "short form"), and an extended argument list (also called the "long form"). These argument blocks are described in greater detail in Section 11.13. For the purposes of this discussion, the short form of the argument block is used in the following examples.

The first word of the LOOKUP/ENTER block contains the name to be assigned to the file. The second word of the argument block contains the file extension. Both the first and second words are SIXBIT names. Words 3 and 4, for present purposes, may be considered null. The four-word argument block for ENTER, assuming we choose "NAME" as the file name and EXT as the file extension, looks as follows:

```
entblock:    SIXBIT/NAME/
             SIXBIT/EXT/
             Ø
             Ø
```

A flow diagram for the I/O sequence is shown in Figure 11-1.

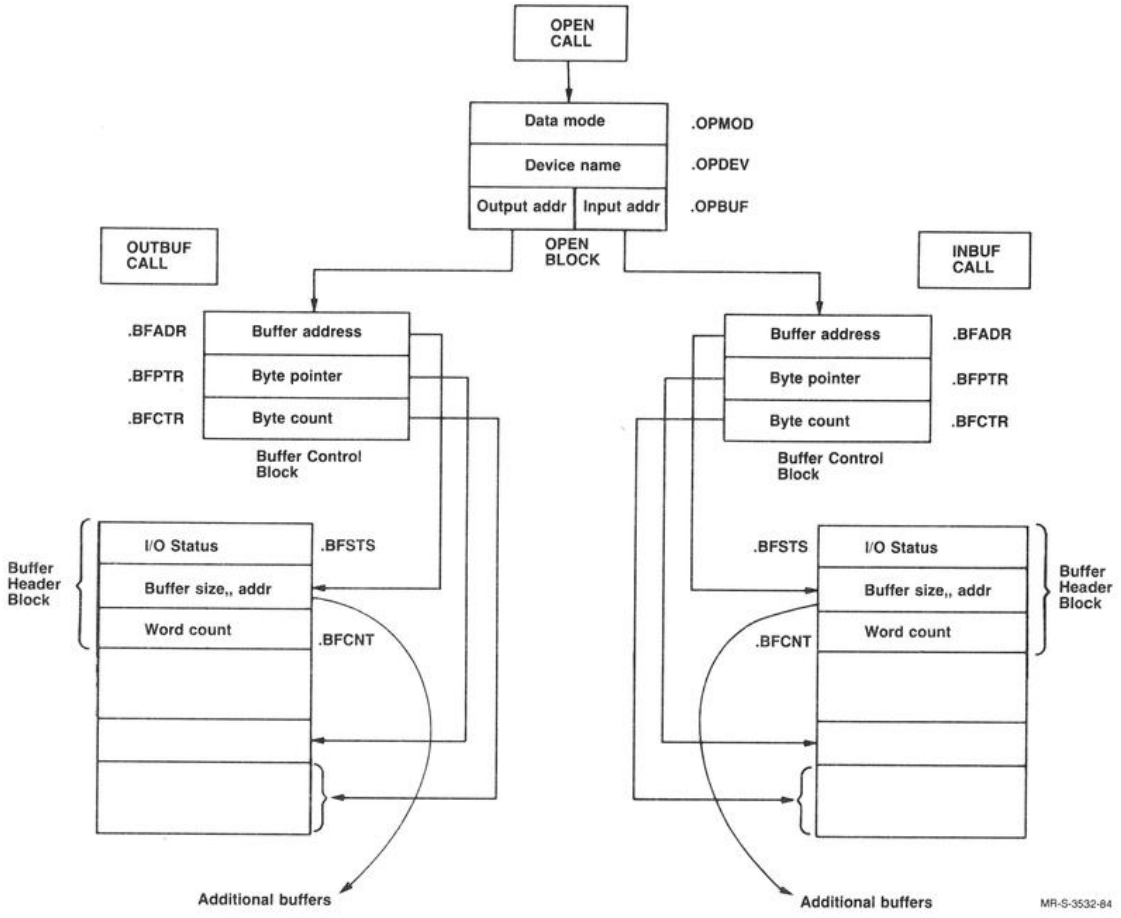


Figure 11-1: Flow Diagram -- I/O Sequence

Here OPEN, OUTBUF, and INBUF are monitor calls while OUTB, OBPTR, and others are arbitrary symbols.

The following example shows a program that writes the file NAME.EXT on disk. The program uses several monitor calls in addition to those already discussed in this chapter, including RESET, CLOSE, OUTSTR, OUT, INCHWL, and EXIT.

```
        TITLE   WRITE1 - Write TTY input to disk
        SUBTTL  HANLEY A. STRAPPMAN 6-27-83/HAS
        SEARCH  UUOSYM

;AC'S
        T1=1                            ;Temp
        P=17                            ;Stack pointer

;MISCL
        D==1                            ;Channel number for disk output
        ESC==33                         ;Use ESCAPE to mark EOF
        PDLSIZ==4                       ;Size of PDL
WRITE1: RESET                           ;Initialize
        MOVE    P,[IOWD PDLSIZ,PDL]     ;Set up PDL
        OPEN    D,DEV                   ;OPEN device
          HALT                          ;Can't
        OUTBUF  D,                      ;This UUO is optional
        ENTER   D,FIL                   ;ENTER the file
          HALT                          ;Can't
        OUTSTR  [ASCIZ /DATA: /]        ;Prompt the user
LOOP:   INCHWL  T1                      ;Input a char from the TTY
        CAIN    T1,ESC                  ;ESCAPE?
        JRST    DONE                    ;Yes
        PUSHJ   P,CO                    ;Output char to disk
        JRST    LOOP

DONE:   CLOSE   D,                      ;CLOSE the disk file
        STATZ   D,IO.ERR                ;Problems during CLOSE?
          HALT                          ;Yes
        RELEAS  D,                      ;This UUO is optional
        EXIT

;Routine to output a char to disk
CO:     SOSGE   OBUF+.BFCTR             ;Room in buffer?
        JRST    CO2                     ;No
        IDPB    T1,OBUF+.BFPTR          ;Yes, store char
        POPJ    P,
CO2:    OUT     D,                      ;Get another buffer
          JRST  CO                      ;Try again
        HALT                            ;Disk error

DEV:    .IOASC                          ;OPEN block
        SIXBIT  /DSK/
        XWD     OBUF,0
OBUF:   BLOCK   3                       ;Ring header
FIL:    .RBEXT                          ;ENTER block
        0
        SIXBIT  /NAME/
        SIXBIT  /EXT/
PDL:    BLOCK   PDLSIZ                  ;Push-down list
        END     WRITE1
```

Note that after OUTBUF and RESET there are no error returns. If CLOSE returns with an error, use GETSTS, STATO, and STATZ to ascertain the error condition.

The program continues to fill the output buffer until, in this case, an ESCape (octal 33) character appears. The byte pointer count in location OBCT in word 2 of the buffer control block maintains a count of the number of bytes remaining in the output buffer.

The OUT monitor call causes the contents of the output buffer to be transferred to the destination device and then clears the buffer except for the buffer header block.

11.7.2 Input (Reading a File)

For input, as for output, you issue an OPEN call and designate a channel over which data is to be passed. Since you are performing input, the third word of OPNBLK must contain, in its right halfword, the location of the first word of the input buffer control block in a manner analogous to output.

For a directory device, the LOOKUP monitor call is then invoked to find the file that is to be read. The format of the LOOKUP call is:

```
LOOKUP   channo,entblock
   error return
normal return
```

where **entblock** is the address of an argument list that specifies the file. This is the same type of argument list as that used for the ENTER call. Note, however, that separate argument lists should be defined for ENTER and LOOKUP calls, because it is possible for the data in the argument block to be changed by the action of these calls.

The following is an example of a program that reads a file:

```
        TITLE   READ1 - Type out a disk file
        SUBTTL  HANLEY A. STRAPPMAN 6-27-83/HAS
        SEARCH  UUOSYM

;AC'S
        T1=1                        ;Temp
        P=17                        ;Stack pointer

;MISCL
        D==1                        ;Channel number for disk
        PDLSIZ==4                   ;Size of PDL

READ1:  RESET                       ;Initialize
        MOVE    P,[IOWD PDLSIZ,PDL];Set up PDL
        OPEN    D,DEV               ;OPEN device
          HALT                      ;Can't
        INBUF   D,                  ;This UUO is optional
        LOOKUP  D,FIL               ;LOOKUP the file
          HALT                      ;Can't
LOOP:   PUSHJ   P,CI                ;Input a char from disk
          EXIT                      ;EOF
        OUTCHR  T1                  ;Output char to TTY
        JRST    LOOP
```

```
;Routine to input a char from disk
;This routine returns noskip if EOF, but otherwise skips
CI:     SOSGE   IBUF+.BFCTR       ;More chars in buffer?
        JRST    CI2               ;No
        ILDB    T1,IBUF+.BFPTR    ;Yes, get one
        JUMPE   T1,CI             ;Ignore nulls
        AOS     (P)               ;Skip return
        POPJ    P,
CI2:    IN      D,                ;Get another buffer
          JRST  CI                ;Try again
        STATZ   D,IO.ERR          ;Error or just EOF?
          HALT                    ;Disk error
        POPJ    P,                ;EOF
DEV:    .IOASC                    ;OPEN block
        SIXBIT  /DSK/
        IBUF
IBUF:   BLOCK   3                 ;Ring header
FIL:    .RBEXT                    ;LOOKUP block
        Ø
        SIXBIT  /NAME/
        SIXBIT  /EXT/
PDL:    BLOCK   PDLSIZ            ;Push-down list
        END     READ1
```

After the LOOKUP, issue an IN monitor call to pass data on the designated channel to the input buffer, thus reading the file. Note that, as with the OUT monitor call, the normal return for IN is the succeeding line with the error return located in the second line following the IN.

Each IN call reads into the buffer one disk block. Subsequent IN calls read sequential blocks starting with the first block in the file. The IN call differs from the OUT call in that each issue of an IN call, including the first in a program, fills a buffer with data as long as there is data left in the file being read. If there is no more data in the file being read, IN takes the error (skip) return.

11.7.3 Writing a File Using FILOP.

The FILOP. UUO can be used in place of the OPEN, OUTBUF, INBUF, and other monitor calls. It performs functions to delete, rename, and append to a file that has been defined in a LOOKUP/ENTER/RENAME argument block. Refer to Chapter 22 for a complete discussion of the FILOP. call.

The following example shows a write operation using FILOP.:

```
        TITLE   WRITE2 - Write TTY input to disk
        SUBTTL  HANLEY A. STRAPPMAN 6-27-83/HAS
        SEARCH  UUOSYM

;AC'S
        T1=1                            ;Temp
        T2=T1+1
        P=17                            ;Stack pointer

;MISCL
        ESC==33                         ;Use ESCAPE to mark EOF
        PDLSIZ==4                       ;Size of PDL

WRITE2: RESET                           ;Initialize
        MOVE    P,[IOWD PDLSIZ,PDL];Set up PDL
        MOVE    T1,[XWD FLOPL,FLOP];ENTER the file
        FILOP.  T1,
         HALT                           ;Can't
        MOVSI   T1,(FO.CHN)             ;Isolate chan number
        AND     T1,FLOP+.FOFNC
        MOVEM   T1,CHAN+.FOFNC
        OUTSTR  [ASCIZ /DATA: /] ;Prompt the user
LOOP:   INCHWL  T1                      ;Input a char from the TTY
        CAIN    T1,ESC                  ;ESCAPE?
        JRST    DONE                    ;Yes
        PUSHJ   P,CO                    ;Output char to disk
        JRST    LOOP

DONE:   MOVEI   T1,.FOCLS               ;Function code
        HRRM    T1,CHAN+.FOFNC
        MOVE    T1,[XWD 1,CHAN]  ;CLOSE the file
        FILOP.  T1,
          HALT                          ;Problems during CLOSE
        EXIT

;Routine to output a char to disk
CO:     SOSGE   OBUF+.BFCTR             ;Room in buffer?
        JRST    CO2                     ;No
        IDPB    T1,OBUF+.BFPTR          ;Yes, store char
        POPJ    P,
CO2:    MOVEI   T2,.FOOUT               ;Function code
        HRRM    T2,CHAN+.FOFNC
        MOVE    T2,[XWD 1,CHAN]  ;Output the buffer
        FILOP.  T2,
          HALT                          ;Disk error
        JRST    CO                      ;Try again

FLOP:   FO.ASC+.FOWRT                   ;Assign channel, write file
        .IOASC
        SIXBIT  /DSK/
        XWD     OBUF,0
        XWD     -1,0                    ;Default number of buffers
        FIL
        0
        FLOPL==.-FLOP                   ;Size of block
CHAN:   BLOCK   1                       ;Channel number
OBUF:   BLOCK   3                       ;Ring header
FIL:    .RBEXT                          ;ENTER block
        0
        SIXBIT  /NAME/
        SIXBIT  /EXT/
PDL:    BLOCK   PDLSIZ                  ;Push-down list
        END     WRITE2
```

11.7.4 Modifying Files (Update Mode)

To modify an existing file, you must be in update mode. As with writing and reading a file, you must first OPEN the channel. You then issue a LOOKUP call and then an ENTER call to the same file, in that order. If you issue an ENTER only, to a file already on disk, the monitor writes a new file. When the file is CLOSED, the old version of the file is deleted from the disk with the new file superseding the old file. Note that to read from one file and write to another, two OPENs referencing two different channels must be used.

LOOKUP and ENTER both reference blocks of identical format; however, the monitor alters the contents of the block when it executes either of these calls. Thus, after a LOOKUP call, the block should usually not be used by an ENTER call. (Refer to Section 11.13.1 and 11.13.2 for more information.)

The monitor maintains pointers, one for each channel in use, designating the block of the file being referenced. You set the pointer on a channel for input or output by using the USETI monitor call (for input) or the USETO monitor call (for output). After LOOKUP and ENTER have been issued for a given file, input and output may be performed on the same channel but there must be two separate buffer header blocks. To reference the first block of data in the file, the LOOKUP call sets the input block pointer to 1 and the ENTER call sets the output block pointer to 1. Each successive IN call causes the input block pointer to be incremented by 1 so the succeeding block will be read by the next IN call. This continues block by block until there are no more blocks left, at which time the EOF bit in the I/O status word is set. The IN attempting to fill an unreserved block will fail, taking the skip return.

Execution of a subsequent OUT call causes the output block pointer to be incremented by 1, which prepares the next OUT call to write the next block of the file.

11.7.5 Block Pointer Positioning

You can control the setting of the input and output data block pointers instead of having them advanced one block at a time with the IN and OUT calls. Using the USETI (User Set Input) and USETO (User Set Output) calls, the pointers may be set to any selected block. (Note that the FILOP. and SUSET. calls can also perform these functions.) The following instruction sets the input block pointer to point to the sixth block from the beginning of the file that is open on channel CHAN:

```
USETI   CHAN,6
```

No input is performed by the USETI call. It simply sets the pointer for a subsequent IN call to read the desired block. A program using the USETO call should have only one buffer for input and output, to simplify keeping track of the block being referenced. Buffered I/O is not recommended for programs doing random access with USETI/USETO.

If a USETI designates a block number larger than that of the last block in the file, the subsequent IN call fails. If a USETI is then issued to an existing block, the EOF bit is cleared by the monitor and input is then again enabled.

Issuing the call USETO CHAN,n with n greater than the number of blocks in the file causes the monitor to allocate any intervening blocks as part of the file. For example, if a file contains 1Ø octal blocks, the call USETO CHAN,6Ø followed by the call OUT CHAN, creates a file of 6Ø octal blocks; the new data is written into the last block and blocks 11-57 (octal) contain zeros.

The USETI and USETO monitor calls do not actually perform any input or output. They change the pointers of the current position of the file. Note that each INPUT and OUTPUT monitor call in your program advances the file. Your program can rewrite or reread the same block by issuing a USETI or USETO before issuing an INPUT or OUTPUT. When a USETI or USETO is executed, the monitor writes all output buffers that your program filled before it changes the pointer to the current position in the file.

Because the monitor reads or writes as many buffers as it can when an INPUT or OUTPUT monitor call is executed, it is difficult for your program to determine which buffer the monitor is processing when a USETI or USETO is performed. Therefore, the INPUT or OUTPUT your program issues following the USETI or USETO might not be reading/writing the buffer that contains the specified block number. A buffer ring consisting of one buffer will read/write the desired block number because the device must stop after each INPUT or OUTPUT. A device having a multiple buffer ring can be stopped after each bufferful of data when your program sets Bit 3Ø (IO.SYN) in the I/O status word. This bit can be set with an INIT, OPEN, FILOP., or SETSTS monitor call. A subsequent USETI or USETO will then specify the correct buffer to be used on the next INPUT or OUTPUT.

If the file protection code does not prevent your job from accessing a file, your program can append data to the last block of an append-only file. You can append data to the file by issuing a USETO to the last block of the file followed by a dummy OUTPUT. The monitor reads the block into the buffer, copies words n+1 through 2ØØ from your buffer into the monitor's buffer, and rewrites the block. On an INBUF monitor call, the monitor sets the byte count. Your program can obtain the current length of the file and the last block by examining the .RBSIZ word of the LOOKUP/ENTER/RENAME block. It is not necessary that your program read the last block of the file before appending to it. Any data that was in the file will not be changed.

When your program appends to the end of a file with append-only protection (protection code 4), the monitor sets the IO.BKT bit in the file status word and performs no output when the following occurs:

- Any block before the last block is written.
- The last block of the file already contains 2ØØ octal words.
- Fewer words are written than the current size of the block.

If the last block of the file is output, the size of the last block becomes 2ØØ (octal) and cannot be appended to, but the entire file can be appended to by creating new blocks.

11.7.6 Super USETI/USETO

The preferred method of performing the functions of the super-USETI/USETO is using the SUSET. monitor call. When using disk packs, there may be a need for your program to read and write data without using a directory hierarchy (such as, for testing a pack in a timesharing environment or for a privileged recovery on any file structure). You must be able to specify individual blocks on a file structure and/or unit without referring to a file. Super-USETI/USETO (or SUSET.) can specify logical blocks within a file structure or physical blocks within a unit. Under certain conditions, USETI and USETO can be used to specify these logical blocks. When the following conditions are true, USETI and USETO reference logical blocks numbers in a file structure, instead of relative blocks within a file:

- When the channel has been initialized with a file structure name (such as, DSKB).
- When no file has been opened on the channel specified in the AC field (that is, no LOOKUP or ENTER has been performed).

Unit referencing occurs when both of the following are true:

- The channel has been initialized with a physical unit name (such as, DPA2) or a logical unit name (such as, DSKC3).
- A file has not been opened on the channel specified in the ac field (that is, no LOOKUP or ENTER has been performed). Super-USETI and super-USETO accept their arguments in the contents of the effective address, because it is possible to have file structures with more than 377777 octal blocks.

SUSET. requires either [1,2] or JACCT privileges, and if you attempt to use it without sufficient privileges, the IO.IMP and IO.BKT bits will be set in the I/O status word.

An output function writes headers and data formats. Only the output immediately following the USETO writes the format. Therefore, to write two IOWDs of format, the following sequence must be used:

```
USETO
OUTPUT
USETO
OUTPUT
```

An INPUT does a read header and data operation if the unit is an RPØ4 or an RPØ6, and it uses an ordinary read operation if the unit is an RPØ2 or RPØ3.

11.8 RECOVERING FROM ERRORS

The OPEN monitor call can fail (taking the non-skip return) if the device is not available to your job.

If the ENTER, LOOKUP, or RENAME call fails, the error code is stored in the LOOKUP/ENTER/RENAME block. This is described in greater detail in Section 11.13. The error codes that can be returned are listed in Section 11.14.

If the IN/INPUT or OUT/OUTPUT call fails, the error code is indicated in the I/O status word (also known as the file status word).

Each channel has an associated I/O status word maintained by the monitor. The setting of certain bits in the I/O status word indicates any errors that may have occurred on the channel. The I/O status word is stored in the same data block with the file by the monitor after you specify the settings of some of the I/O status bits and the data mode in your argument block to the INIT, OPEN, or FILOP. call to open the I/O channel. You can set I/O status bits that control the data transmission. The data mode that you specify for the channel will also be stored in the I/O status word.

The I/O status word appears as the right half-word returned by the GETSTS call. Bits 18 through 29 contain the I/O status bits. The I/O status bits are usually set by the monitor to indicate the state of the file after a transfer. The I/O error bits are Bits 18 through 22 (IO.ERR). Your program can set these bits using the SETSTS call. However, the I/O status bits returned by the monitor are different for each type of device. They are described, therefore, in the chapters describing specific types of devices.

The data mode is stored in the I/O status word in Bits 30 through 35. The data modes are described in Section 11.4.

The EOF (end-of-file) bit (Bit 22) is set to 1 if an IN call tries to read past the EOF. You can use the following instruction to check the status of error bits in the I/O status word:

```
STATO     CHAN,x
```

The STATO call causes a skip if any of the bits in the I/O status word for the channel CHAN that are masked by the right half word value x are set to 1. The error return for the IN call should therefore contain the following code:

```
STATO     CHAN,IO.EOF
  JRST    ERROR           ;Not end-of-file
```

Before continuing as before, the remaining error bits of the I/O status word could also be checked using STATO or STATZ monitor calls

Extended error codes are used to extend the limited set of error bits allowed in the I/O status word. On an extended error, all of the error bits in IO.ERR are set. If your program encounters such an error, it should proceed to use the .DFRES function of the DEVOP. call to determine the exact nature of the error.

11.9 USING BUFFERED I/O

Buffered I/O allows the monitor to overlap I/O transfers with other program operations. The buffered data modes are ASCII, ASCII Line, Packed Image, Byte, Image, Image Binary, and Binary. These modes use a buffer ring for both input and output.

Your program specifies the location of a buffer control block, which the monitor sets up. This buffer control block contains pointers to the current buffer and the current byte in the buffer. A buffer ring structure is shown in Figure 11-2.

A program using buffered I/O can become larger and larger because of repeated INBUF and OUTBUF monitor calls. Before an OPEN call, the current pointers to the buffers should be saved from the buffer ring header. After the OPEN call, the pointers may be restored. Growth of the user area may be thus avoided by saving and restoring Word Ø of the buffer ring header before and after an OPEN call, or by setting .JBFF in the job data area to an appropriate value.

The buffer control block is either three or four words long. The fourth word is present only for MPX-controlled devices. When the buffer ring is initialized, the buffer control block points to the first buffer in the buffer ring. Thereafter, the buffer control block points to the current buffer in the buffer ring. The buffers are linked to form an endless ring. The buffer header block of each buffer points to the next buffer in the ring. The monitor sets up and maintains the buffer header blocks.

Your program need only specify the location of the buffer control block. This specification is made in an OPEN, INIT, or FILOP. monitor call. However, if your program should decide at a later time to change the location of the ring control block, it can issue the MVHDR. monitor call.

Your program can specify the number of buffers to be contained in a buffer ring, or the monitor will set up the default number of buffers.

Figure 11-2 shows a buffer ring containing a buffer control block and three buffers.

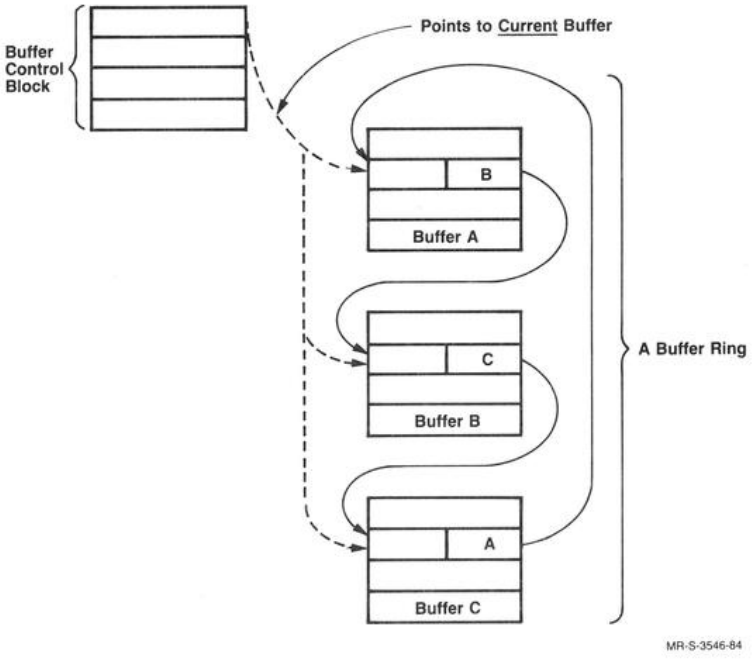


Figure 11-2: The Buffer Structure

Normally, buffers are filled and emptied in the order in which they are placed in the buffer ring. However, your program can specify a deviation from this normal buffering scheme. To deviate from the normal scheme, your program specifies the buffer's address in the IN, INPUT, OUT, or OUTPUT monitor call.

Buffered I/O is performed in the same manner for all devices except those connected to an MPX channel. (For more information on buffered I/O, refer to the sections 11.9.3 to 11.9.7)

Note that when a buffer ring is initialized, the monitor places the buffers at the first free location, whose address is contained in .JBFF in the job data area. After placement of the buffers, the monitor updates .JBFF to contain the first free location after the buffers. Your program must use FILOP. when placing buffers in a non-zero section. .JBFF is available for programs running in Section Ø only.

11.9.1 Initializing a Buffer Ring

In buffered I/O, buffers allow overlap of a program's execution and I/O transfers. While your program processes data in one buffer, the monitor fills or empties another buffer. The I/O transfer also overlaps the processing of other user jobs running on the same system.

Normally, the monitor sets up buffers at the end of your low segment, unless you tell it otherwise. However, your program may also set up the buffers. An arbitrary number of buffers may be used for a given device or file. These buffers are linked to form a buffer ring structure.

The first three words in each buffer are not used to hold data. Instead, they hold information pertaining to the buffer. This information includes: the address of the next buffer in the buffer ring, status bits concerning the I/O device being used, and a bit indicating whether the buffer has been filled. These three words are referred to as the buffer header and are not part of the data on the device.

Associated with every buffer ring is another group of three words (four for MPX-controlled devices). These three words are called a buffer control block. The buffer control block contains information that is examined to determine whether your program can access the buffers in the ring. Your program must reserve space for the buffer control block, and your program must inform the monitor of its location. Buffer rings, buffer headers, and buffer control blocks are described in detail in the following sections.

Typically, the monitor sets information in the buffer control block every time your program issues a monitor call that requests that a new buffer be made available to your program. As your program works through a buffer, your program usually increments the byte pointer and decrements the byte counter. Each time the counter expires, your program can execute another monitor call asking for another buffer.

To perform buffered input/output, your program should use the following monitor calls:

- RESET initializes your program.
- INIT, FILOP., and OPEN initialize devices.
- INBUF, OUTBUF, and FILOP., initialize a buffer ring (optional).
- LOOKUP, ENTER, FILOP., USETI, USETO, and SUSET. select a file and a block within a file.
- IN, INPUT, OUTPUT, FILOP., and OUT transmit data.
- CLOSE and FILOP. close a file.
- RELEASE and FILOP. terminate device activity.
- RESET and RESDV. forcibly terminate device activity.
- EXIT terminates your program.

11.9.2 The INBUF and OUTBUF Monitor Calls

The INBUF and OUTBUF monitor calls set up an input and output buffer ring with a specified number of buffers, starting at the location pointed to by .JBFF. Alternatively, the INBUF and OUTBUF monitor calls can be left out of your program; in which case, the monitor sets up one of the following:

- A ring of two buffers for non-disk devices.
- Or a ring of n buffers for disk devices. You can set the value of n by using the SET DEFAULT BUFFERS monitor command or corresponding SETUUO monitor call. If no default is specified, the monitor uses the system default, an installation parameter that can be defined with MONGEN (usually 6).

The calling sequences for the INBUF and OUTBUF monitor calls are as follows:

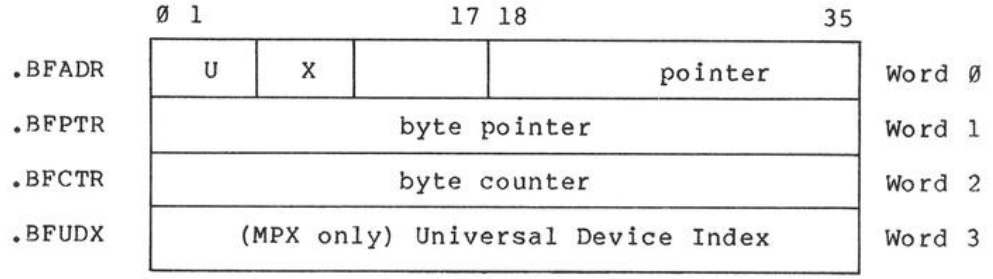
```
INBUF channo,n          OUTBUF channo,n
return                  return
```

where channo is the channel number associated with the device. You can initialize an I/O channel using OPEN, INIT, or FILOP.

n specifies the number of buffers in the buffer ring. If n is zero or omitted, the monitor sets up the default number of buffers. FILOP. may be used to specify the number of input and/or output buffers.

11.9.3 The Buffer Control Block

The buffer control block is initialized when your program executes its first OPEN, INIT, or FILOP. monitor call specifying a buffered I/O data mode. Your program specifies the location of this control block in either the OPEN, INIT, or FILOP. monitor call, and the monitor sets up its contents. You can change this address, and therefore change the control block, using the MVHDR. monitor call. This call changes the monitor pointer to a buffer control block from one address to another. The new control block is at the new address; the monitor does not "move" the control block, but merely looks for it in the new place you specified with the MVHDR. call. The format of the buffer control block is shown below:



The information in the buffer control block is described below.

In Word Ø the symbol U, Bit Ø, (BF.VBR) indicates the use bit, which is set if there has been any input to or output from the buffer ring.

X (BF.IBC), Bit 1 is set to inhibit the clearing of output buffers. To enforce this, your program must set this bit as well as UU.IBC in the OPEN argument block.

pointer (.BFADR) is the address of the current buffer in the buffer ring. This half-word points to the second word (Word 1) of the current buffer.

In Word 1 the **byte pointer** (.BFPTR) points to the byte within the current buffer that contains the next input or output data. The byte size is determined by the data mode.

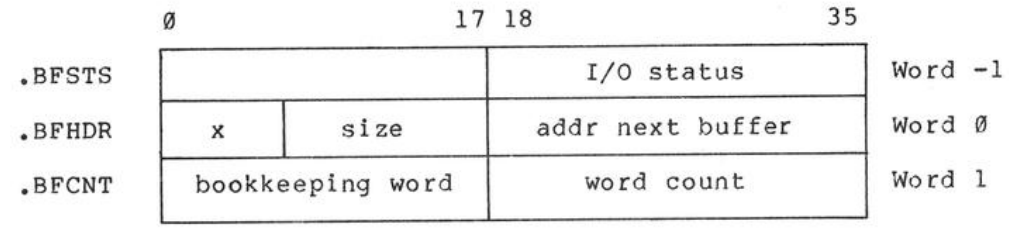
In Word 2 the **byte counter** (.BFCTR) is the count of the number of bytes remaining in the current buffer on input and the number of bytes for which there is still room on output.

In Word 3 the **Universal Device Index** (.BFUDX) is present only for devices connected to an MPX channel. This word identifies which of the devices connected to the MPX channel is the current device (that is, the device to which the current buffer applies). Your program supplies this word in the buffer control block on output. The monitor supplies it on input. Selective input, therefore, is not possible on MPX-controlled channels.

A user program cannot use the same buffer control block for both input and output. Also, the same buffer control block cannot be used for more than one I/O device at a time (except for MPX-controlled devices, refer to Section 11.9.7). Therefore, users cannot use the same buffer control block for simultaneous input and output, and only one buffer ring can be associated with each buffer control block.

11.9.4 The Buffer Header Block

There is one buffer header block for each buffer in a buffer ring. The monitor maintains the contents of the buffer header blocks, and all buffer linkages. The format of a buffer header block is:



(Note that UUOSYM.MAC defines .BFSTS as Ø.) The information maintained by the monitor in the buffer header block is described below.

In Word Ø, the **I/O status** (.BFSTS) contains the status of the file when the monitor advances to the next buffer in the ring. This word contains the errors received while working with the file. User programs should not be written to reference this word; instead, use the GETSTS, STATO, or STATZ monitor calls.

In Word 1, **x** (BF.IOU) is the buffer's use bit. This bit is a flag that indicates that the buffer contains active data. If the buffer contains data, the bit is set. If the buffer is empty, this bit is off.

BF.IOU is set by the monitor when the buffer is full on output or has been filled on input (that is, if the use bit = Ø, the buffer is available to the filler, if it is 1, the buffer is available to the emptier). The setting and clearing of the buffer's use bit prevents the monitor and your program from interfering with each other by attempting to use the same buffer simultaneously. Your program does not advance to the next buffer; the monitor advances the buffer ring on the execution of certain monitor calls. Note that the monitor sets and clears the buffer's use bit; your program should never change its state.

size (BF.SIZ) is the size, in words, of the data area in the buffer, plus one. The size of the data area is dependent on the type of device being used.

addr next buffer (BF.NBA) is the address of the next buffer in the ring. (This address is the second word of the next buffer.)

In Word 2, bookkeeping word is reserved for bookkeeping purposes; the exact purpose depends on the device used and the data mode specified. When a device connected to an MPX channel is specified, this word contains the Universal Device Index associated with that device. When using DECtape, it is the next block number on the tape for the next record of the file.

word count (.BFCNT) is reserved for a count of the number of words that actually contain data. When using byte mode, this value is equal to the number of bytes that actually contain data.

11.9.5 Using Buffered Input

Using buffered input can speed your program's execution. In buffered input, the monitor sometimes fills the buffers for your program while the program is performing other tasks; thus buffers can be filled ahead and be ready when the program requests more data.

Figure 11-3 is a flowchart showing the monitor's handling of buffered input.

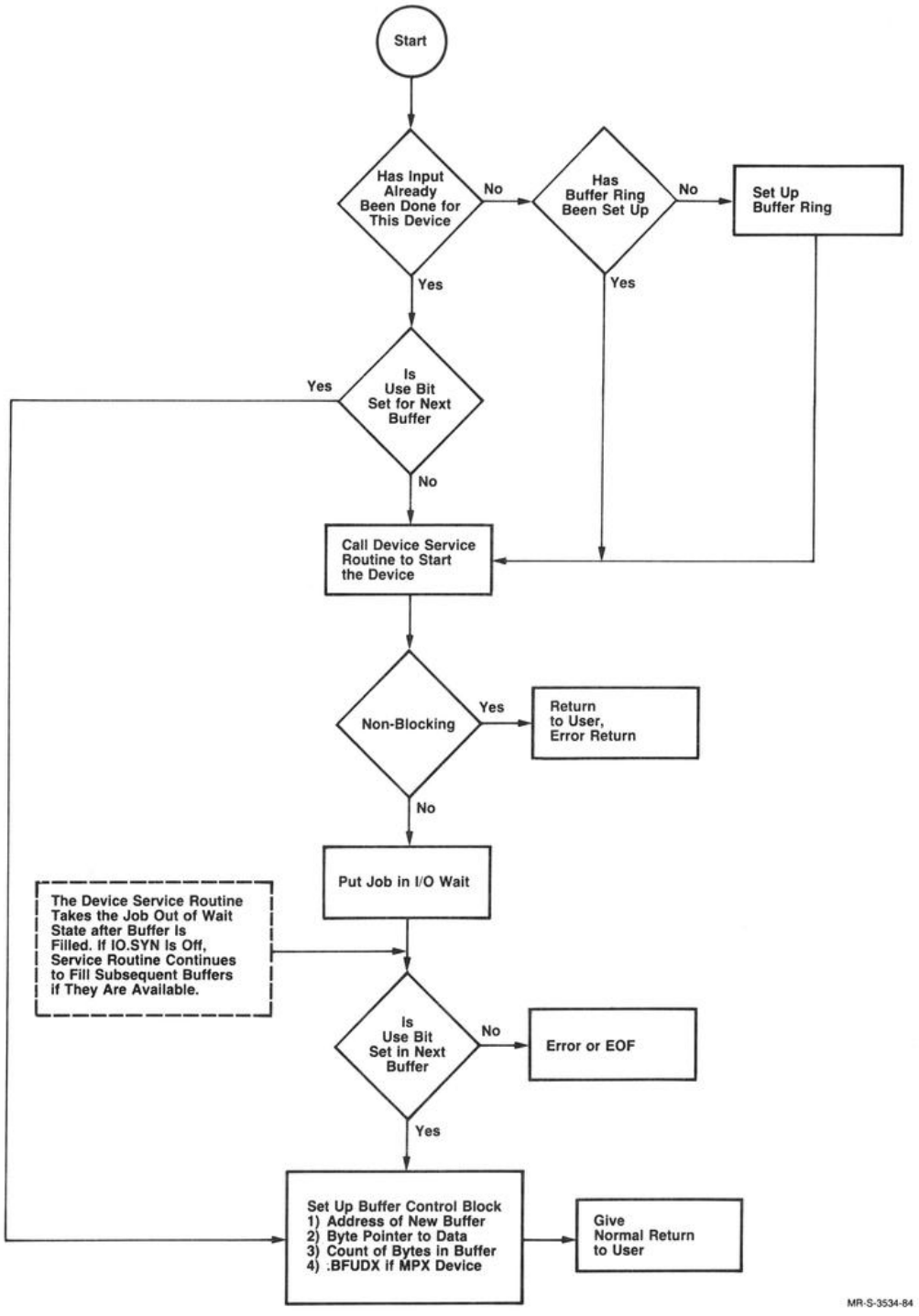


Figure 11-3: Flowchart for Buffered Input

To use buffered input, your program should:

1. Use the OPEN, INIT, or FILOP. monitor call to specify buffered mode, a device, and the location of the buffer control block for the input file.

2. Optionally, use the INBUF or FILOP. monitor call to specify the number of buffers in the ring.

3. Use IN or INPUT monitor calls to read from the file.

There are four ways to do buffered input:

- In synchronous mode, blocking and non-blocking I/O.

- In asynchronous mode (default), blocking and non-blocking I/O.

For all of these methods, your program requests the next bufferful of data by using an IN, INPUT, or FILOP. monitor call.

11.9.5.1 Normal Buffered Input - In normal (asynchronous blocking) buffered input, the monitor takes the following actions on each IN or INPUT monitor call:

1. Checks the use bit to determine whether to put your job in wait state. When the data is ready, the monitor advances to that buffer and gives a success or error return based on the contents .BFSTS in that buffer.

2. Advances the buffer ring pointer to the next buffer.

3. Updates the buffer control block, including the pointer to the current buffer, the byte pointer to the data, and the byte count.

4. Returns control to your program.

Note that if the next buffer is not ready, your job is put into a wait state until the buffer is ready. The advantage of using buffered input is that after the monitor returns control to your program, the monitor continues to fill empty buffers in the ring. The monitor does this while your program is running. Therefore, subsequent INs or INPUTs in your program have a chance of finding the next buffer ready and avoiding the need to be put into the wait state.

11.9.5.2 Synchronous Buffered Input - You may want to prevent the monitor from filling buffers ahead, perhaps because error recovery procedures are in progress or you are doing USETIs to specify exact blocks. This is called synchronous buffered input.

To use synchronous buffered input, use the SETSTS monitor call to set the IO.SYN bit in the I/O status word for the device. Then, when the recovery procedure is completed, you can clear this bit to resume filling ahead. The monitor may be filling buffers ahead of your program. After using SETSTS, your program should check the buffer use bit to determine the current buffer.

You can also suspend your program's execution temporarily (for example, to recover from an I/O error) by using the WAIT monitor call. This call causes your program's execution to wait for completion of any I/O operations that are in progress on a given channel.

11.9.5.3 Nonblocking Buffered Input - You may want the monitor to continue executing your program, rather than place your job in a wait state, if the next needed buffer is not ready. For example, you may want your program to service several devices. This is called nonblocking buffered input.

To use nonblocking buffered input, your program must set the bit UU.AIO (asynchronous I/O) in the first word of the OPEN argument block. Your program must use the IN or FILOP. monitor call (not the INPUT monitor call) for nonblocking buffered input.

In nonblocking buffered input mode, the monitor takes the error return (with no error code in the I/O status word) from an IN monitor call if the buffer is not ready. Therefore your program can determine whether the input was done, and can proceed appropriately.

To determine whether error bits are set on the return from an IN monitor call, use the GETSTS, STATZ, FILOP., or STATO monitor call. If no error bits are set, then there are no buffers containing data ready and your program can perform other operations not associated with the same device (such as computations or I/O for other devices).

Your program can periodically retry the IN call, to see if a buffer is ready. Your program can also be written to respond to an input-done software interrupt when a buffer is ready (see Chapter 7), or to wake from a hibernating state when the buffer is ready (see the HIBER UUO in Chapter 22).

11.9.6 Using Buffered Output

Using buffered output can speed your program's execution. In buffered output, the monitor writes the buffer's data as soon as the buffer is filled. Thus your program need not determine when a buffer is ready for output.

Figure 11-4 is a flowchart showing the monitor's handling of buffered output.

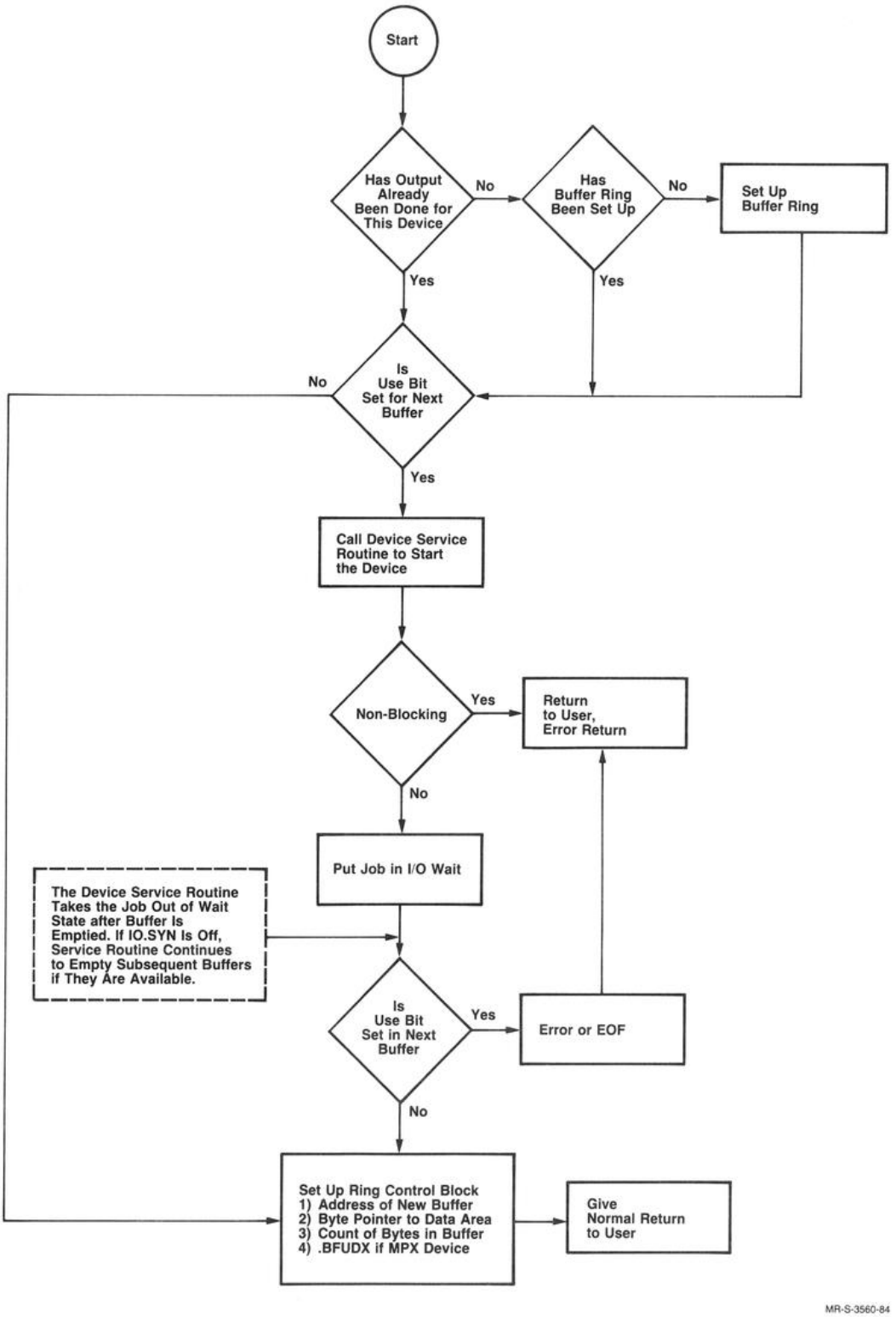


Figure 11-4: Flowchart for Buffered Output

To use buffered output, your program should:

1. Use the OPEN, INIT, or FILOP. monitor call to specify buffered mode, a device, and the location of the buffer control block for the output file.

2. Optionally, use the OUTBUF or FILOP. monitor call to specify the number of buffers in the ring.

3. Issue a "dummy" OUT or OUTPUT monitor call to initialize the buffer ring. This is normally transparent to the program and does not require special coding.

4. Use OUT or OUTPUT monitor calls to write to the file.

If the BF.IOU bit in the buffer control block and the IO.UWC bit in the I/O status word are both cleared, the monitor assumes that the current buffer is empty. The monitor then keeps track of the number of bytes in the buffer as it is filled. This value is stored in the .BFCTR word of the buffer control block.

If the IO.UWC bit in the I/O status word is set, the monitor assumes that your program has already computed the number of words in the buffer. If you are in .IOBYT mode, the monitor assumes that your program has already computed the number of bytes in the buffer. The monitor then sets the BF.IOU (use) bit in the buffer control block and starts the device needed to empty the buffer.

11.9.6.1 Normal Buffered Output - In normal buffered output, the monitor takes the following actions on each OUT, OUTPUT, or FILOP. output monitor call:

1. Checks to make sure the next buffer in the ring is empty. If not, the monitor places the job in a wait state and starts the device needed to empty the buffer.

2. Advances the buffer ring pointer to the next buffer.

3. Updates the buffer control block, including the pointer to the current buffer, the byte pointer to the data, and the item byte count.

4. Returns control to your program.

Note that if the next buffer is not ready, your job is put into a wait state.

11.9.6.2 Synchronous Buffered Output - You may want to prevent the monitor from filling buffers ahead, perhaps because error recovery procedures are in progress. This is called synchronous buffered output.

To use synchronous buffered output, use the SETSTS monitor call to set the IO.SYN bit in the I/O status word for the device. Then, when the recovery procedure is completed, you can clear this bit to resume buffering ahead.

You can also suspend your program's execution temporarily (for example, to recover from an I/O error) by using the WAIT monitor call. This call causes your program's execution to wait for completion of any I/O operations that are in progress on a given channel.

11.9.6.3 Nonblocking Buffered Output - You may want the monitor to continue executing your program (rather than place your job in a wait state) even if the next needed buffer is not ready. For example, you may want your program to be allowed to service other devices. This is called nonblocking buffered output.

To use nonblocking buffered output, your program must set the bit UU.AIO in the first word of the OPEN argument block. Your program must use the OUT or FILOP. monitor call (and **not** the OUTPUT monitor call) for nonblocking buffered output.

In nonblocking buffered output mode, the monitor takes the error return (with no error code in the I/O status word) from an OUT or FILOP. monitor call if the buffer is not ready. Therefore your program can determine whether the output was done, and can proceed appropriately.

To determine whether error bits are set on the return from an OUT monitor call, use the GETSTS, STATZ, or STATO monitor call. If no bits are set, your program can perform other operations not associated with the same device (such as computations or I/O for other devices).

Your program can periodically retry the OUT or FILOP. call, to see if the buffer is ready. Your program can also be written to respond to an output-done software interrupt when the buffer is ready (see Chapter 6), or to wake from a hibernating state when the buffer is ready (see Chapter 22).

11.9.7 Buffered I/O for MPX-Controlled Devices

The monitor recognizes that I/O is for an MPX-controlled device because you OPENed MPX (or a logical name for it). The buffer control block must be 4 words long; the last word will contain a Universal Device Index.

For input, MPX-controlled devices use a conventional buffer ring (as described above). For output, they use a special buffer structure that consists of a free chain (for the MPX channel) and one device chain for each device on the channel.

The free chain is a series of buffers in which each buffer points to the next. For each buffer in the series, the BF.NBA field of the buffer header contains the address of the next buffer; BF.NBA in the last buffer header contains Ø.

The .BFADR word of the buffer control block points to the current buffer in the free chain, so that there is a continuous chain from the buffer control block to the last buffer in the free chain.

The buffer control block and the buffers in the free chain are in user core. The device data block (DDB) for the MPX channel is in monitor core.

Each device chain consists of a DDB for the device and one or more buffers linked to the DDB. The DDB points to the first buffer in the device chain. Each buffer (in its .BFADR word) points to the next buffer; the last buffer has Ø in the right half of .BFADR.

As output proceeds in your program, the monitor handles output for MPX-controlled devices as follows:

1. The first OUT or OUTPUT monitor call to the MPX channel is the "dummy" call. The monitor creates the free chain for the MPX channel.

2. For each call to a device on the MPX channel, the monitor moves a buffer from the top of the free chain to the bottom of the device chain. This is done by updating the .BFADR address in the buffer control block to point to the second buffer in the free chain, which then becomes the top buffer in the free chain. The BF.NBA field in the moved buffer is zeroed, because it is to be the end of the device chain. The address of the moved buffer is placed in BF.NBA for the old end buffer in the device chain, so that the chain is extended to include the moved buffer.

3. When a buffer is emptied (written to its device), the monitor moves the buffer back to the bottom of the free chain. This is done by updating the pointers in both the device chain and the free chain.

The following pages contain figures and explanations showing how the monitor handles buffered output for an MPX channel.

Recall that the first OUT or OUTPUT call for the MPX channel is the "dummy" call, and merely creates the free chain for the channel. Figure 11-5 shows a 4-buffer chain that is created after the dummy call.

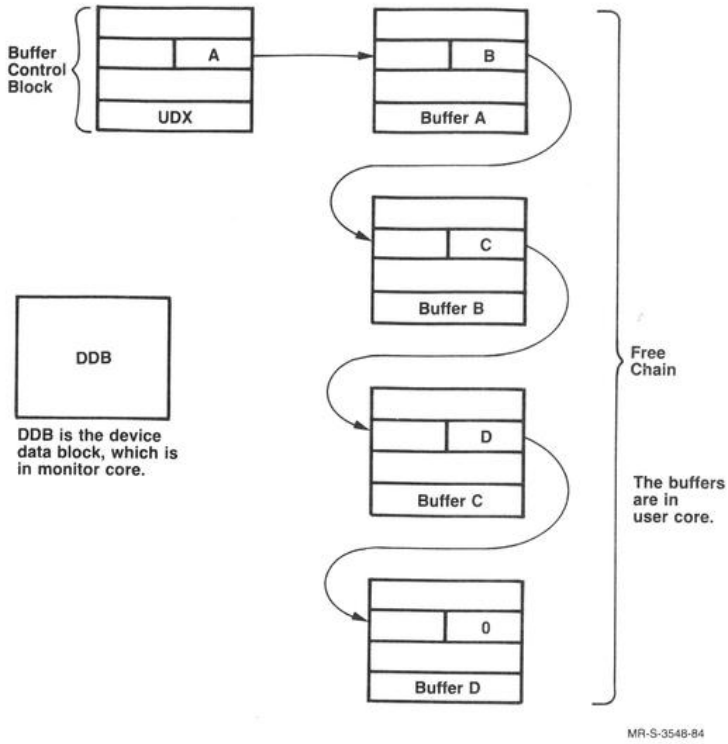


Figure 11-5: MPX Buffer Control Block with Four Buffers

The next OUT or OUTPUT call for an MPX-controlled device on the channel moves a buffer from the top of the free chain to the bottom of the (formerly empty) device chain for that device. Figure 11-6 shows this situation.

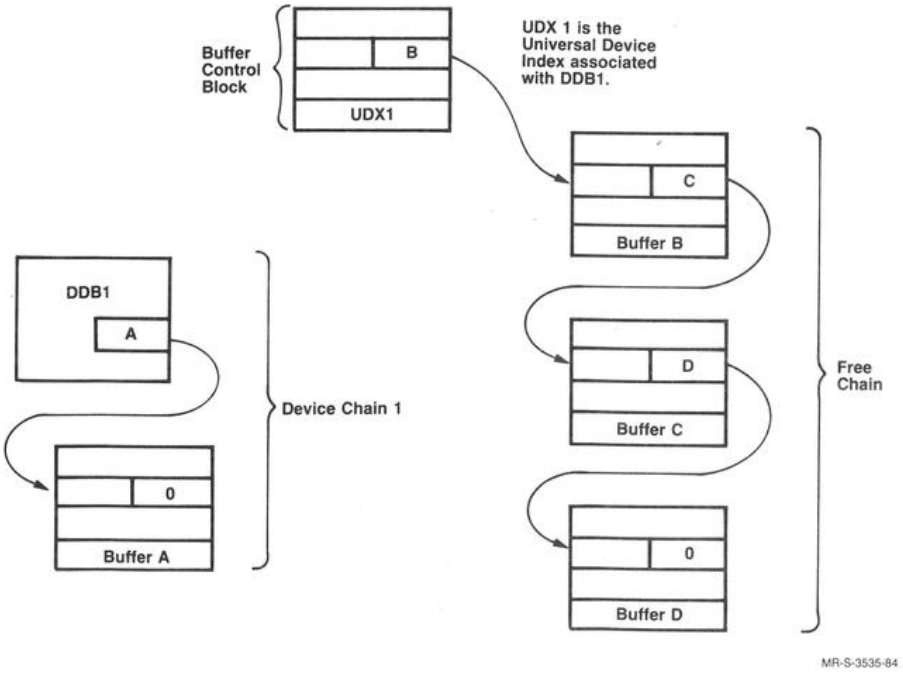


Figure 11-6: One Buffer in One Device Chain

The first OUT or OUTPUT call for a different MPX-controlled device on the channel (a device with a different Universal Device Index) moves a buffer from the top of the free chain to the bottom of the (formerly empty) device chain for that device. Figure 11-7 shows this situation, in which there are two device chains.

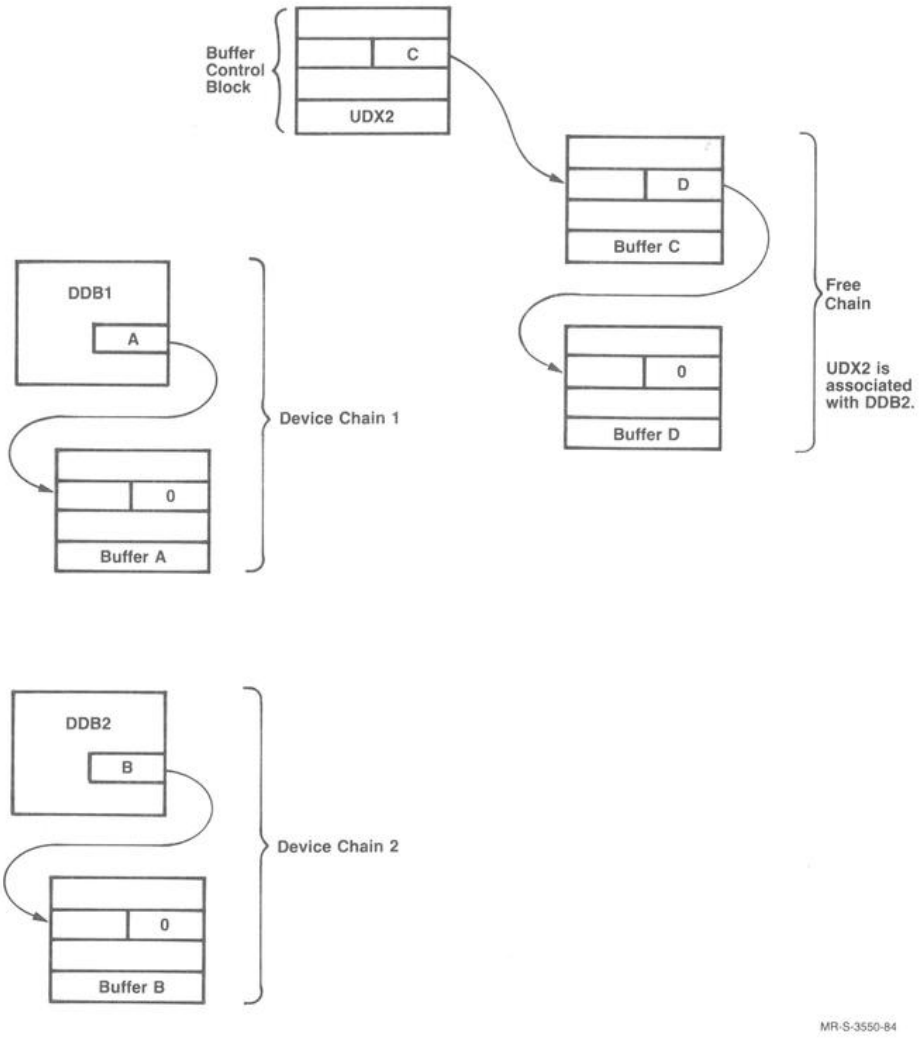


Figure 11-7: One Buffer in Each of Two Device Chains

Another OUT or OUTPUT call for a device that already has a device chain moves a buffer from the top of the free chain to the bottom of the device chain for that device. Figure 11-8 shows this situation, in which multiple device chains have multiple buffers.

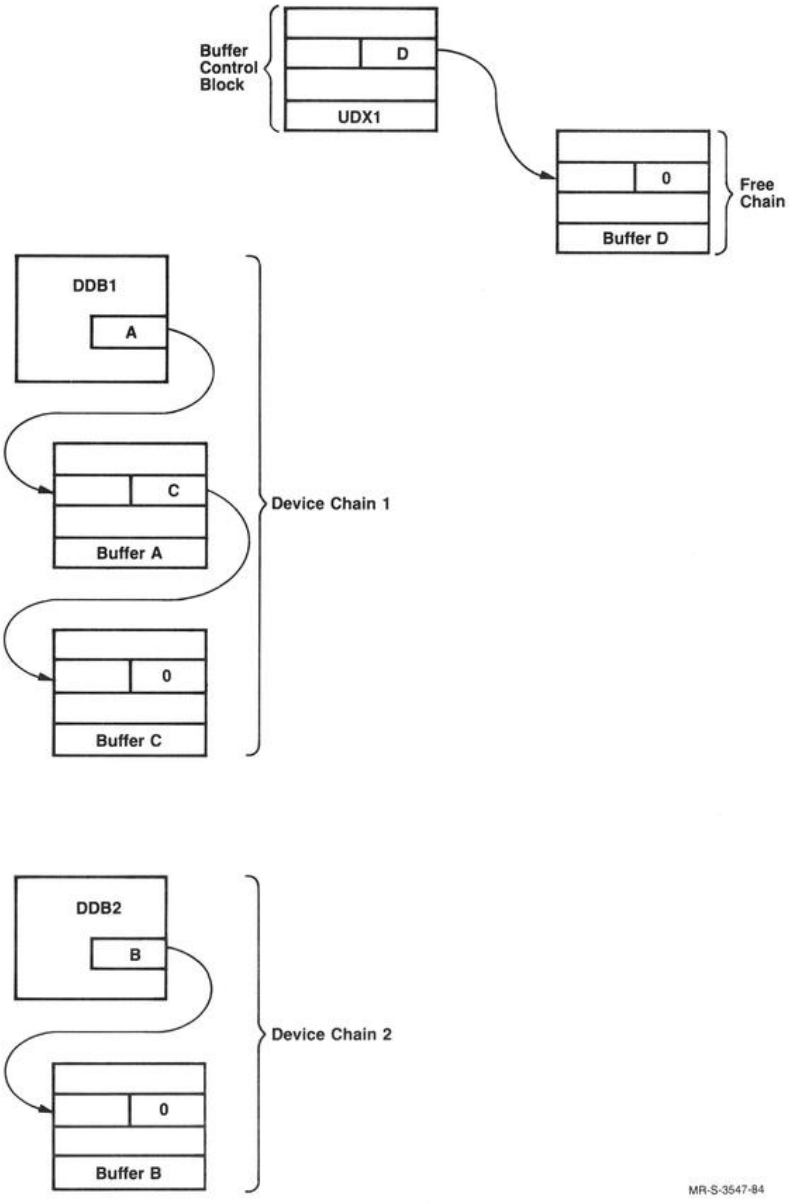


Figure 11-8: Multiple Buffers in Multiple Device Chains

When the monitor has emptied a buffer in a device chain, it returns the buffer to the bottom of the free chain. Figure 11-9 shows this situation, in which one the buffers for a device has been moved from the device chain to the bottom of the free chain.

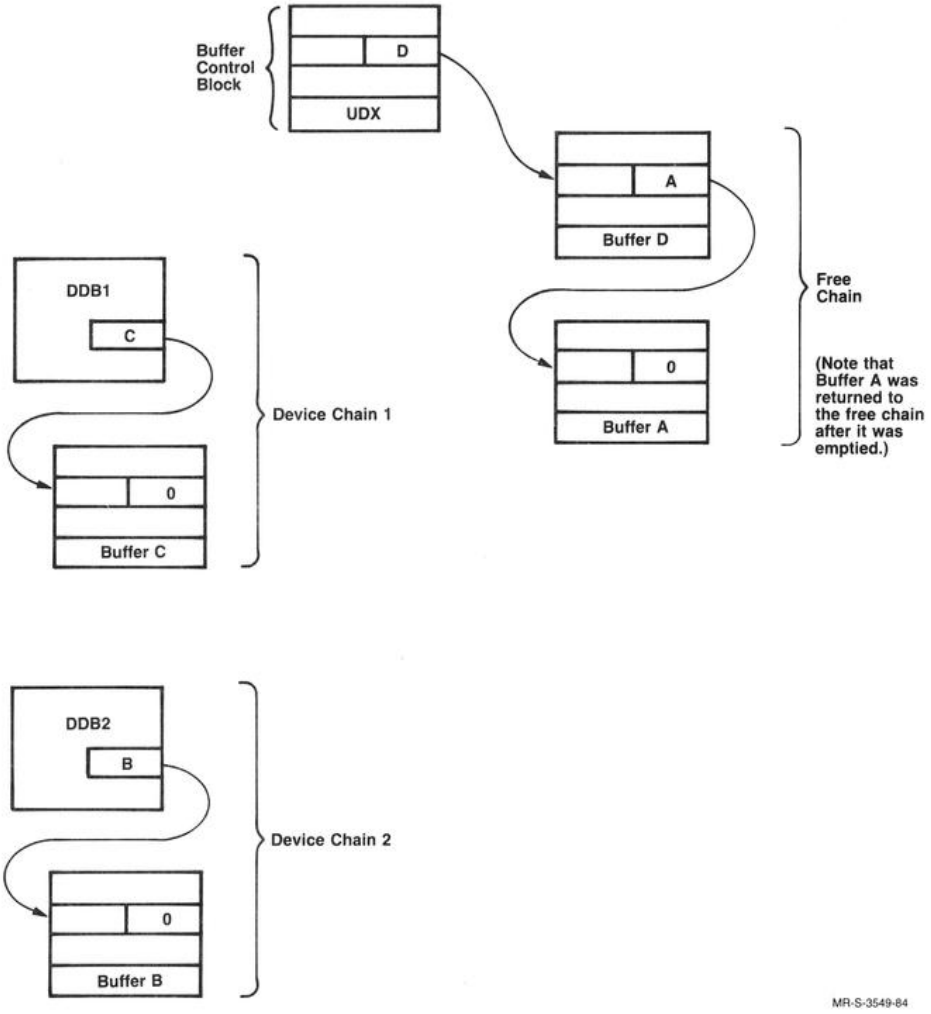


Figure 11-9: One Buffer Moved Back to Free Chain

11.9.8 Generating Your Own Buffers

Your program can generate its own buffers instead of allowing the monitor to generate them. You might want to do this, for example, if your buffers must use a nonstandard byte size.

The following example shows how to set up buffers for an input file. This code sequence is similar in its performance to the INBUF monitor call. In the example, the device is MTAØ; the undefined symbol BYTSIZ gives the byte size for the buffers; the undefined symbol SIZE gives the size of the buffers; the number of buffers in the ring is 2.

```
;This example shows how to set up a 2-buffer input buffer ring.

GO:     OPEN    ICHN,OPNBLK         ;Initialize the channel
          JRST  ERROR               ;Not available
        MOVE    T1,[BF.VBR+BUF1+.BFHDR]
        MOVEM   T1,MAGBUF+.BFADR
        MOVESI  T1,(POINT BYTSIZ)
        MOVEM   T1,MAGBUF+.BFPTR
        JRST    CONTIN              ;On to something else

OPNBLK: BLOCK   1                   ;For flags and status bits
        SIXBIT  /MTAØ/              ;Device name
        XWD     Ø,MAGBUF            ;No output,,input ring header

MAGBUF: BLOCK   3

BUF1:   BLOCK   1                   ;For file status
        XWD     SIZE+1,BUF2+.BFHDR  ;Size+1,,next buffer
        BLOCK   1                   ;For word count
        BLOCK   SIZE                ;Space for the buffer

BUF2:   BLOCK   1                   ;For file status
        XWD     SIZE+1,BUF1+.BFHDR  ;Size+1,,next buffer
        BLOCK   1                   ;For word count
        BLOCK   SIZE                ;Space for the buffer

CONTIN:                             ;Something else
```

If you have a program with a specialized memory management scheme and want to specify the buffer locations, you would want to generate your own buffers. The following example shows how this situation could be handled.

```
     T1=1
     T2=2

GO:     OPEN    ICHN,OPNBLK         ;OPEN file
          JRST  ERROR               ;Not available
        MOVEI   T1,OPNBLK           ;Find number and
        DEVSIZ  T1                  ; size of
          JRST  ERROR               ; default buffers
        HLRZ    T2,T1               ;Compute words needed
        HRRZS   T1                  ; for size and number
        IMULI   T1,(T2)             ; of buffers
        PUSHJ   P,GETMEM            ;Call your memory allocator
                                    ; with words to get in T1,
                                    ;returns address of block in T1
          JRST  ERROR               ;Can't get?
        PUSH    P,.JBFF             ;Save current first free
        MOVEM   T1,.JBFF            ;Default number of buffers
        INBUF   T1,Ø                ;Make monitor put buffers
                                    ; where you want
        POP     P,.JBFF             ;Restore .JBFF now that
                                    ; buffers are allocated
        .
        .
        .
        .
```

The following example demonstrates the general principles of using buffered I/O:

```
TITLE   Buffered Input/Output Example for TOPS-10

COMMENT *

This program demonstrates the basic principles of buffered input and
output by reading the ASCII input file INPUT:INPUT.IN and copying it
to the ASCII output file OUTPUT:OUTPUT.OUT.

Note that the logical names INPUT: and OUTPUT: must be assigned by the
ASSIGN command.  These logical names can be assigned to any devices
that support ASCII line mode.  The monitor will ignore the LOOKUP and
ENTER calls if the devices do not support directories.

END COMMENT *

SEARCH  UUOSYM                          ;Use standard definitions

SALL                                    ;Keep the program neat

;Definitions for registers and channels

T1=1                                    ;For scratch
C=10                                    ;For storing a character
J=15                                    ;For JSPing around

ICHN==1                                 ;Input channel number
OCHN==2                                 ;Output channel number

;Macro for general messages

DEFINE  ERROR(TEXT)<
        JRST    [OUTSTR [ASCIZ |?'TEXT
|]                                      ;Output message to TTY:
        JRST    MONRT]                  ;And back to monitor mode
>                                       ;End of macro definition

;Here on entry to initialize program.

BUFENT: JFCL                            ;In case of CCL entry (ignore)
        RESET                           ;Reset any I/O, .JBFF, etc.
                                        ;(In case of CTRL/C start)

        OPEN    ICHN,INDEV              ;Open input device
          ERROR <CAN'T OPEN LOGICAL DEVICE INPUT:>
        LOOKUP  ICHN,INFIL              ;Find input file
          ERROR <CAN'T LOOKUP INPUT FILE INPUT.IN>
        OPEN    OCHN,OUDEV              ;Open output device
          ERROR <CAN'T OPEN LOGICAL DEVICE OUTPUT:>
        ENTER   OCHN,OUFIL              ;Create output file
          ERROR <CAN'T ENTER OUTPUT FILE OUTPUT.OUT>
COMMENT *

Here we could optionally set up buffer rings using the INBUF and
OUTBUF monitor calls.  Instead, we'll let the monitor do it for us on
the first IN and OUT monitor calls. This is normal.
```

```
END COMMENT *

;Here's the main I/O loop to transfer the file.

IO:     JSP     J,GETBYT            ;Read a byte of input
          JRST  EOF                 ;End of input file
        JSP     J,PUTBYT            ;Write the byte
        JRST    IO                  ;Back for next byte

;Here on end of input file.

EOF:    RELEAS  ICHN,               ;Let input device and channel go
        CLOSE   OCHN,               ;CLOSE output file
                                    ;(Writing last buffers)
        STATZ   OCHN,IO.ERR         ;Any last errors?
          JRST  OUERR               ;Yes, complain to user
        RELEAS  OCHN,               ;No, release output device

;Here to return to monitor mode.
;If user continues, restart.

MONRT:  MONRT.                      ;Back to monitor mode
        JRST    BUFENT              ;User said continue

COMMENT *

The following routines are the basic low-level buffered I/O  routines.
Note  that both GETBYT and PUTBYT are self-initializing.  On the first
call to each, the input and output byte  counts  are  Ø  (set  by  the
monitor on the OPEN).

GETBYT falls into the IN call to set up the default  number  of  input
buffers  and  starts filling them.  Control returns to GETBYT when the
first buffer is full.

PUTBYT falls into the OUT call (the so-called "dummy" OUT) to  set  up
the  default  number  of  output buffers.  Then it returns so that the
program can fill and output the buffers.

END COMMENT *

;GETBYT - Routine to read 7-bit ASCII input data.

GETBYT: SOSGE   INCNT               ;Any chars in input buffer?
          JRST  GETBUF              ;No, must read next buffer
        ILDB    C,INPTR             ;Yes, return char in C
        JUMPN   C,1(J)              ;Successful return is skip
        JRST    GETBYT              ;Null char, throw away

GETBUF: IN      ICHN,               ;Advance to next input buffer
        JRST    GETBYT              ;And back for next character
        GETSTS  ICHN,T1             ;Input error
        TRNE    T1,IO.EOF           ;Was it end-of-file?
          JRST  (J)                 ;Ok, not a real error
        ERROR   <I/O ERROR ON INPUT>
```

```
;PUTBYT - Routine to write 7-bit ASCII output data.

PUTBYT: SOSGE   OUCNT               ;Any room in current output buffer?
          JRST  PUTBUF              ;No, write out and get next
        IDPB    C,OUPTR             ;Yes, put it in output buffer
        JRST    (J)                 ;Return for more (note nonskip)

PUTBUF: OUT     OCHN,               ;Write out this buffer
        JRST    PUTBYT              ;And start filling next
OUERR:  ERROR   <I/O ERROR ON OUTPUT>

;Here all data blocks are defined.
;First the input and output OPEN blocks:

INDEV:  .IOASL                      ;ASCII line mode
        SIXBIT  /INPUT/             ;Input device name
        XWD     Ø,INHDR             ;Address of input buffer
                                    ;Control block

OUDEV:  .IOASL                      ;ASCII line mode
        SIXBIT  /OUTPUT/            ;Output device name
        XWD     OUHDR,Ø             ;Address of output buffer
                                    ;Control block

;NOW THE LOOKUP/ENTER BLOCKS:

INFIL:  SIXBIT  /INPUT/             ;Input file name
        SIXBIT  /IN/                ;Input extension
        BLOCK   1                   ;Protection, mode, creation date
        BLOCK   1                   ;PPN or path number

OUFIL:  SIXBIT  /OUTPUT/            ;Output file name
        SIXBIT  /OUT/               ;Output extension
        BLOCK   1                   ;Protection, mode, creation date
        BLOCK   1                   ;PPN or path pointer

;Next the buffer control blocks. (The monitor will build the
; buffers on the first IN and OUT calls.):

INHDR:  BLOCK   1                   ;Input buffer control block
INPTR:  BLOCK   1                   ;Input buffer ring byte pointer
INCNT:  BLOCK   1                   ;Input buffer ring byte count

OUHDR:  BLOCK   1                   ;Output buffer control block
OUPTR:  BLOCK   1                   ;Output buffer ring byte pointer
OUCNT:  BLOCK   1                   ;Output buffer ring byte count

        END     BUFENT
```

11.10 CLOSING A FILE

To close a file, use the CLOSE or FILOP. monitor calls. These calls end I/O operations for the file and ensure that all data input or output is completed.

11.10.1 Maintaining File Integrity

Ordinarily, the integrity of a disk file is not assured until you perform a CLOSE operation on the file. For instance, if the system crashes when writing a disk file, the entire disk file to date is lost. The entire current file would also be lost if a system crash occurred during an update-mode writing of the file. This happens because an update-mode writing that extends the file allocates new disk blocks to the file; therefore, the file's RIB must also be re-written. However, after you perform the CLOSE, the monitor guarantees the file's integrity even across a system crash, unless something destroys the physical file structure. There are, however, two methods you can use to assure file integrity while actively using the file:

- Using the FILOP. .FOURB function. This FILOP. function writes the complete file to disk, and updates the file's RIB on disk as though you had performed a CLOSE. The .FOURB function acts as a checkpoint operation for a disk file. After the FILOP. .FOURB, the file is guaranteed on disk and will survive a system crash or any halt, such as a <CTRL/C>. Programs that perform journaling operations use this function to save user input in a separate file. This journal file is saved on disk if an unexpected halt occurs. Later, you can recall this file and use it to restore previous input and re-execute commands.

  You should note that the monitor performs an OUT monitor call for you when you use this procedure. This positions the buffer header byte pointer at the beginning of a word, no matter where the byte pointer was before the .FOURB call. If the byte pointer is in the last word of a buffer, then the header points at the next buffer in the ring after the .FOURB call. As a result, the disk file may contain embedded null bytes of data for each of the checkpoint operations executed.

- Using either the FILOP. .FORRC function, or setting the UU.RRC bit in the OPEN monitor call. If you periodically issue the FILOP. monitor call with the .FORRC function, the monitor will checkpoint the file if the RIB has changed since the last .FORRC call. The program may enable a PSI program interrupt whenever a disk file RIB has changed (interrupt condition PS.RRC).

  If you set the UU.RRC flag when you issue the OPEN monitor call, the monitor automatically checkpoints your file when you write enough data to cause a change in the RIB.

11.11 RELEASING A DEVICE

To release a device, use the RELEASE or FILOP. monitor call. These calls return the device and its channel to the monitor's pool.

11.12 STOPPING A PROGRAM

To stop execution of a program, use the EXIT monitor call. This call terminates execution of the program, but leaves the program in your user memory so that it can be restarted.

Most programs can also be stopped by the CTRL/C command. If the program is waiting for terminal input, type one CTRL/C. If not, type two CTRL/Cs.

11.13 THE LOOKUP/ENTER/RENAME ARGUMENT BLOCKS

As discussed in Section 11.7, the LOOKUP, ENTER, and RENAME monitor calls accept argument lists in the identical formats. There are two formats you can use to supply arguments to these calls. The short form allows you to accomplish the call by specifying a minimal amount of information. This form is used in Section 11.7 and following to illustrate the general sequence of calls needed to accomplish I/O. The short form of the argument block is described in detail in Section 11.13.1.

The long form of the argument block (also called the "extended argument list") is used to specify information for disk files. This format of the LOOKUP/ENTER/RENAME argument block can also be used to read the RIB (Retrieval Information Block) of the disk file. For the purpose of providing completely device-independent I/O code, the extended argument list can be used for I/O on any device. The information that is not applicable to each device is simply ignored. The extended argument list is described in detail in Section 11.13.2.

11.13.1 The Short Form of the Argument List

The LOOKUP/ENTER/RENAME argument list may take the following form. The short form of the argument list must always be 4 words long. The following list shows the contents of each word in the argument list. The arguments are denoted by the following symbols for each of the three monitor calls, LOOKUP, ENTER, and RENAME:

A	signifies an argument that your program must supply.
AØ	signifies an optional argument. If your program does not supply the contents, the monitor uses a default value.
V	signifies a value that is only returned by the monitor after the call is completed.

The short form of the argument list for LOOKUP, ENTER, and RENAME monitor calls is:

Word	LOOKUP	ENTER	RENAME	Contents
Ø	A	A	A	File name in SIXBIT.
1	A	A	A	Bits Ø-17: file extension in SIXBIT.
	V	AØ	AØ	Bits 18-2Ø: high-order three bits of the creation date.
	V	AØ	AØ	Bits 21-35: access date.
	V	V	V	Bits 18-35: on an error return from the call, the error code is stored here. Refer to Section 11.14.
2	V	AØ	A	Bits Ø-8: protection code.
	V	V	V	Bits 9-12: data mode of file when it was created.
	V	AØ	A	Bits 13-23: creation time in minutes since midnight.
	V	AØ	A	Bits 24-35: low-order twelve bits of creation date.
3	A	A	A	Word 3 on input is: PPN or path pointer. A path pointer takes the form: Ø,,addr where addr is the address of the path block. Refer to the PATH. UUO in Chapter 22.
3	V	-	-	Word 3 is returned as: Bits Ø-17: the LOOKUP call returns the length of the file in the left half as number of words expressed as a negative number. The file size is expressed as a positive number when the file contains more than 128K words.

11.13.2 The Extended Argument List

The long form of the argument block for LOOKUP, ENTER, and RENAME monitor calls allows you to specify more information about the file. You can also maintain greater control over your I/O request using flags provided in this argument list.

The extended argument list is signified by placing a zero in the left half of the first word of the argument list, and the length of the argument list in the right half. The total length must be at least 3 words. .RBMAX is the maximum number of words (5Ø octal) that the block may contain. The system ignores any value larger than this.

The argument list allows your program to supply information that is passed to the monitor. If your program includes an illegal argument, the monitor ignores that information and returns the default value of the incorrect argument. Each word of the extended argument list is described below. The following symbols are used in the description to denote the applicability of each argument to each of the monitor calls, LOOKUP, ENTER, and RENAME.

The symbols in the columns are:

- **A** = an argument (supplied by either a privileged or unprivileged program) and returned by the monitor as a value.
- **AØ** = an argument like A, except that a Ø argument causes the monitor to substitute a default value.
- **Al** = an argument if supplied by a privileged program; if supplied by an unprivileged program, it is ignored.
- **V** = the value returned by the monitor cannot be set even by a privileged program; the monitor will ignore the argument.

Word	Symbol	LOOKUP	ENTER	RENAME	Contents
Ø	.RBCNT	A	A	A	The count of the number of arguments that follow. Left half: unused, must be zero. Right half: flags + number of arguments following this.

Where the flags can be:

Bits	Symbol	Meaning
18	RB.NSE	If set on an ENTER, that ENTER is a non-superseding ENTER. If your program specifies an existent file name, that file will not be superseded. The monitor will give the error return and will return error code 4 (ERAEF%) in addr+3 of the argument block.
19	RB.DSL	(Don't Search LIB) During a LOOKUP, if the file is not found in the default path, the monitor would, by default, proceed to search LIB. Failing that, if /SYS is enabled, the monitor would search SYS (and, with /NEW enabled, NEW). Setting RB.DSL inhibits these actions. If the file is not found in the default path, the monitor will immediately return error ERFNF% and no further searching will take place. The default path will always be scanned. RB.DSL does not effect the use of SFD scanning.

Bits	Symbol	Meaning
20	RB.AUL	(Allow Updates in LIB) By default, if a LOOKUP finds a file on device LIB, the monitor will not allow a subsequent ENTER or RENAME. This action is intended to prevent the user from accidentally modifying the wrong file. If RB.AUL is set, however, the user's actions are regarded as deliberate, and the subsequent ENTER or RENAME will be allowed. Note that RB.AUL must be on at the time of the LOOKUP, and that this bit is meaningless for ENTER and RENAME calls.
21	RB.NLB	(No Load Balancing) When a user ENTERs a file on a structure that consists of multiple disk units, the monitor creates the file on the unit with the most available space, by default. However, if the user has a channel open for another file on the same structure, the monitor attempts to create the new file on a different unit in the structure, regardless of the unit that has the most space. The two files are created on different units in the same structure to ensure that I/O to the structure is evenly balanced. If the program sets RB.NLB, the new file is not forced to a different unit than the originally opened file, suppressing the load balancing action. Under RB.NLB, each file is created on the unit in the structure that has the most available space.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
1	.RBPPN	A0	A0	A0	A PPN or a pointer to a path block. For a path pointer, the left half contains zero, and the right half contains the address of the path block. For a description of a path block, refer to the PATH. UUO in Chapter 22. The PPN is for the user file directory in which the file is to be LOOKed UP, ENTERed, or RENAMEd. To LOOKUP a UFD, .RBPPN must contain 1,,1 (indicating the MFD). The search defaults to LIB and SYS only if the directory path was not specified.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
2	.RBNAM	A	A	A	The SIXBIT file name, left-justified with trailing nulls. If the Master File Directory or a User File Directory is being LOOKed UP, ENTERed, or RENAMEd on this call, this location contains the directory name. The argument can be Ø on a RENAME or a LOOKUP and ENTER if pathological names are in use, in which case the file will be deleted.
3	.RBEXT	A	A	A	Bits Ø-17: The SIXBIT file extension, left-justified with trailing nulls. Although file extensions are optional, null extensions are discouraged because they convey no information.
		V	AØ	AØ	Bits 18-2Ø (RB.CRX): The high-order three bits of the 15-bit creation date. The system updates the creation date only when you write additional blocks to the file. For instance, altering the last block of the file would not result in an updated creation date.
		V	AØ	AØ	Bits 21-35 (RB.ACD): The access date.
		V	V	V	Bits 18-35: If the UUO fails, an error code is returned in the right half of this word. Refer to Section 11.14 for a list of the error codes.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
4	.RBPRV	V	AØ	A	Bits Ø-8: Protection code (RB.PRV).
		V	V	A	Bits 9-12: Data mode in which the file was created (RB.MOD).
		V	AØ	A	Bits 13-23: Creation time in minutes since midnight (RB.CRT). The system updates the creation time only when you write additional blocks to the file.
		V	AØ	A	Bits 24-35: The low-order 12 bits of the 15-bit creation date in standard format (RB.CRD).
5	.RBSIZ	V	V	V	The written length of the file, in words. If your program sets this value, the monitor ignores it.
6	.RBVER	V	A	A	The octal version number of the file, same as .JBVER.
7	.RBSPL	V	A	A	The file name to be used to label the output from a spooled device. The file name is specified on an ENTER to the spooled device, or it is Ø if an ENTER has not been performed.
1Ø	.RBEST	V	A	A	The estimated length of the file, in positive number of blocks. On the execution of an ENTER call, the monitor uses this value as the number of blocks to allocate for the file. If the requested number of blocks cannot be allocated, partial allocation is performed, and the normal return is taken. .RBALC always contains the actual number of blocks allocated.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
11	.RBALC	V	A	A	The number of contiguous 128-word blocks allocated to a file when an ENTER or RENAME call is performed. The number of blocks includes the RIBs of the file and is equivalent to the last block number of the file. .RBALC equal to Ø does not change the allocation of the file. All of the data blocks can be deallocated by superseding the file and performing no output before the CLOSE. This argument can be used to allocate additional space onto the end of the file, deallocate previously allocated but unwritten space, or truncate written blocks. The smallest unit of disk space that the monitor can allocate is a cluster of 2ØØ-word blocks. Typically, small devices use a cluster size of 1 block. If the number of blocks allocated is not equal to the last block of a cluster, the monitor will round up; thereby adding a few more blocks than the user requested. If the monitor cannot allocate the specified number of blocks, then the partial allocation error (error code 17) will be returned; however, your program may still write the file. To create a file of prespecified length, your program should perform an extended ENTER with .RBEST set and .RBALC cleared. To create a file of prespecified length with contiguous blocks, your program should perform an extended ENTER with .RBEST cleared and .RBALC set. After an ENTER, .RBALC will contain the accurate allocated file length.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
12	.RBPOS	V	A	A	The logical block number of the first allocated block for a new group of clusters appended to the file. The logical block number is specified with respect to the entire file structure, beginning with block number Ø. Combined with the DSKCHR call, this feature allows your program to allocate a file with respect to tracks and cylinders for maximum efficiency when the program is executed.
13	.RBUFW	V	V	V	Determines the disk drive that the file was written on; the format is as follows: Bits Ø-9: Reserved for DIGITAL. Bits 1Ø-17 (RB.UNI): Unit(s) that have written the file (Bit 17 set = drive Ø, Bit 16 set = drive 1, and so forth). Bits 18-2Ø (RB.CON): The controller number (Ø = A, 1 = B, and so forth) of the controller that last wrote the file. Bits 21-35 (RB.APR): The serial number of the CPU that last wrote the file.
14	.RBNCA	A	A	A	Reserved for customer definition; does not require privileges.
15	.RBMTA	V	A1	A1	A 36-bit tape label, if the file has been put onto magnetic tape.
16	.RBDEV	V	V	V	The logical name of the unit on which the file is located.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
17	.RBSTS	V	A1	A1	The I/O status word (.RBSTS). Left half: The status of the UFD. Right half: The status of the file. Refer to .RBSTS bit definitions in the next table for the bit definitions of this word.
2Ø	.RBELB	V	V	V	The logical block number within the unit on which the first data error or search error (IO.DTE) occurred, as opposed to the block within the file structure. This value is set in the RIB by the monitor when a CLOSE is executed and the hardware has detected a hard parity error or a search error while reading or writing the file. Device errors, checksum errors, and redundancy errors are not stored in this location. Any data you place in this location will be ignored, but the following bits may be returned by the monitor: Bits Ø-2 (RB.EVR): Error type: bad version block number. Bit 3 (RB.ETO): Error type: other (not data or search error). Bit 4 (RB.ETD): Error type: data (parity or hard ECC). Bit 5 (RB.ETS): Error type: search or header compare. Bits 3-8 (RB.ETM): Mask of all error type bits. Bits 9-35 (RB.EBN): Number (within unit) of first bad block.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
21	.RBEUN	V	V	V	Left half: The logical unit number within the file structure on which the last bad region was detected. Right half: The number of bad blocks in the last detected bad region. The bad region may extend beyond the file. This argument is ignored, and a value is returned. Bits Ø-8 (RB.ENB): Number of contiguous bad blocks. Bits 1Ø-17 (RB.EUN): Unit number within controller; bit 1Ø = unit 7, bit 17 = unit Ø. Bits 18-2Ø (RB.EKN): Controller number. Bits 21-35 (RB.ECN): CPU number.
22	.RBQTF	V	A1	A1	Contains the logged-in quota This quota is the maximum number of data and RIB blocks that can be in this structure's directory while the user is logged-in. The UFD and the UFD's RIB are not included in this count.
22	.RBTYP	V	A	A	Contains file type and flags. This word is used by system programs (such as FORTRAN), and its format is determined by the application in which it is used. Refer to UUOSYM for a complete description of this word.
23	.RBQTO	V	A1	A1	Contains the logged-out quota (meaningful for the UFD only). This quota is the maximum number of data and RIB blocks that can be left in this structure's directory after you log out. LOGOUT requires that the user must be below this quota to log out.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
23	.RBBSZ	V	A	A	Contains byte and record length information. System programs (such as FORTRAN) use this word, and its format is determined by the application in which it is used. Refer to UUOSYM for a complete description of this word.
24	.RBQTR	V	A1	A1	Reserved quota (applies only to UFDs). This information is not completely implemented by the monitor. (Meaningful for UFD only.)
24	.RBRSZ	V	A	A	Contains record and block size information. System programs (such as FORTRAN) use this word; its format is determined by the application in which you use it. Refer to UUOSYM for a complete description of this word.
25	.RBUSD	V	A1	A1	Contains the number of data and RIB blocks allocated to files in this structure's directory when the owner last logged off (meaningful for the UFD only). LOGIN reads this word so that it does not have to LOOKUP all files to set up the number of written blocks. LOGIN sets Bit Ø (RP.LOG) of the file status word (see below), and LOGOUT clears it to indicate whether LOGOUT has stored the quantity.
25	.RBFFB	V	A	A	Contains first free byte and application-specific information. This word is used by system programs (such as FORTRAN.) Refer to UUOSYM for a complete description of this word.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
26	.RBAUT	V	A1	A1	The PPN of the job creating or superseding the file, as opposed to the owner of the file. Usually the author and the owner are the same. Only when a file is created in a different directory are these different.
30	.RBIDT	V	A1	A1	BACKUP'S incremental date and time in UFD.
31	.RBPCA	V	A1	A1	Privileged argument reserved for customer definition.
32	.RBUFD	V	V	V	The logical block number in the file structure of the RIB for the UFD in which the file appears.
33	.RBFLR	V	V	V	The relative block number of the file to which the first pointer of this RIB points; this is used for multiple RIBs (for example, 0 = prime RIB).
34	.RBXRA	V	V	V	The extended RIB address (that is, the logical unit number and the cluster address of the next RIB in a multiple RIB file).
35	.RBTIM	V	V	V	The internal creation and time of the file, in the universal date-time format.
36	.RBLAD	V	A1	A1	The last accounting date. Valid only for UFDs.
37	.RBDED	V	A1	A1	The directory expiration date. For disk directories, this is valid only for UFDs. For magtape, this is valid only for labelled tapes and refers to the expiration date of the file.
40	.RBACT	A	A1	A1	Account string word 1, in ASCII. This is non-zero on an ENTER, and is not valid for UFDs.
41	.RBAC2	A	A1	A1	Account string word 2, in ASCII.

Word	Symbol	LOOKUP	CREATE/ SUPERSEDE	UPDATE/ RENAME	Contents
42	.RBAC3	A	Al	Al	Account string word 3, in ASCII
43	.RBAC4	A	Al	Al	Account string word 4, in ASCII.
44	.RBAC5	A	Al	Al	Account string word 5, in ASCII.
45	.RBAC6	A	Al	Al	Account string word 6, in ASCII.
46	.RBAC7	A	Al	Al	Account string word 7, in ASCII.
47	.RBAC8	A	Al	Al	Account string word 8, in ASCIZ.

A null byte terminates the string.

Bits	Symbol	Meaning
Ø	RP.LOG	Set if the user is logged in. LOGIN sets this bit; LOGOUT clears it.
5	RP.CHG	Set if some file has changed in this UFD since the last BACKUP.
7B11	RP.UER	All UFD errors.
9	RP.UCE	Set if any file in this UFD has had a software checksum error or a redundancy check error.
1Ø	RP.UWE	Set if any file in this UFD has had a hard data error while writing.
11	RP.URE	Set if any file in this UFD has had a hard data error while reading.
18	RP.DIR	Set if the file is a directory file; this protects the system from a user trying to modify a directory file. The protection error is given if the extension UFD is specified on an ENTER or RENAME and this bit is not set.
19	RP.NDL	If set, the file cannot be deleted, renamed, or superseded, even by a privileged program.
2Ø	RP.DMP	Set if this is an unprocessed crash file. This bit is set on CRASH.EXE files and used by the CRSCPY program.
21	RP.NFS	Set if the file should not be dumped by disk backup programs because certain files (for example, SWAP.SYS, SAT.SYS) contain no useful data to write on the tape.
22	RP.ABC	Set if the file always has bad checksums (because the monitor never recomputes a checksum) for example, SWAP.SYS, SAT.SYS.

Bits	Symbol	Meaning
23	RP.CBS	If set, RP.CMP (Bit 26) is set on entry to UFD compressor.
24	RP.ABU	If set, disk backup programs should always dump this file.
25	RP.NQC	If set, the file is a non-quota checked file, and it is not billed to the user's disk quota.
26	RP.CMP	If set, the UFD is being compressed.
27	RP.FCE	If set, the file has a software checksum error or a redundancy check error (the IO.IMP bit has been set).
28	RP.FWE	If set, the file has had a hard data error while writing. An entry is made in the BAT block so that the bad region is not reused.
29	RP.FRE	If set, the file has had a hard data error while reading. An entry is made in the BAT block so that the bad region is not reused.
30	RP.RMS	If set, this file was created by RMS (the Record Management Service).
31	RP.PAL	If set, this is a preallocated file. This bit is set when you preallocate a file (using FILOP.) but the file has not been created yet (that is, the file is null). This bit is cleared when data is in the file.
32	RP.BFA	If set, the file is bad because of a tape read error during a restore.
33	RP.CRH	If set, the file was closed after a crash.
35	RP.BDA	If set, the file has been marked as bad by a damage assessment program.
715	RP.ERR	All file errors.

11.14 ERROR CODES

Error codes are restricted to a maximum of 15 bits to eliminate problems when recovering from an error in a file with a zero creation date. The following error codes are returned from ENTER, LOOKUP, RENAME, RUN, GETSEG, MERGE., FILOP., and SAVE. calls. For more information, refer to the appropriate call in Chapter 22.

Code	Symbol	Error
0	ERFNF%	The specified file was not found, a null file name was specified, file names do not match on an update operation, or RENAME after a LOOKUP failed. For a FILOP., this error code is returned if the specified device cannot perform I/O in the direction indicated.
1	ERIPP%	The UFD does not exist on the specified file structure (incorrect PPN).

Code	Symbol	Error
2	ERPRT%	Protection failure, or directory is full for a DECtape.
3	ERFBM%	File being modified (ENTER, RENAME).
4	ERAEF%	The specified file already exists (RENAME, FILOP.), a different file name was specified (ENTER after a LOOKUP), the file was superseded (on a non-superseding ENTER).
5	ERISU%	Inclusion of an illegal sequence of monitor calls (such as a RENAME with no preceding LOOKUP or ENTER, or a LOOKUP after an ENTER). For example, the system returns this error code if you attempt to RENAME a file on device LIB without setting FILOP. bit RB.AUL first.
6	ERTRN%	One of the following errors occurred: o Transmission, device, or data error (RUN, GETSEG, MERGE. only). o Hardware-detected device or data error detected while reading the UFD's RIB or the file's RIB. o Software-detected data inconsistency error detected while reading the UFD's RIB or the file's RIB.
7	ERNSF%	File is not in executable format (RUN, GETSEG, MERGE. only).
10	ERNEC%	Not enough core available to load the file (RUN, GETSEG, MERGE., SAVE. only).
11	ERDNA%	Device not available (RUN, GETSEG, FILOP., SAVE., MERGE.).
12	ERNSD%	No such device (RUN, GETSEG, MERGE., SAVE.). For FILOP., this error code is returned if an open function fails to assign the device.
13	ERILU%	Illegal monitor call for FILOP. or GETSEG.
14	ERNRM%	There is no room on this file structure or the disk space quota was exceeded (an overdraw quota is not considered).
15	ERWLK%	A write-lock error occurred. The program cannot write on this device.
16	ERNET%	Not enough table space is available in the monitor's free core.
17	ERPOA%	Partial allocation only.
20	ERBNF%	Block not free at allocated position (ENTER, RENAME).
21	ERCSD%	Cannot supersede a directory (ENTER).
22	ERDNE%	Cannot delete a directory that is not empty (RENAME).
23	ERSNF%	The sub-file directory was not found (some SFD in the specified path was not found).
24	ERSLE%	The search list is empty (a LOOKUP or an ENTER was performed on the generic device DSK and the search list was empty).
25	ERLVL%	You cannot create an SFD nested deeper than the maximum allowed level of nesting.
26	ERNCE%	No file structure in the job's search list has both the no-create bit and the write-lock bit equal to zero and has the UFD or SFD specified by the default or explicit path (ENTER on the generic device DSK only).

Code	Symbol	Error
27	ERSNS%	The program performed a GETSEG from a locked low segment to a high segment that was not a dormant, active, or idle segment. The segment was not on the swapping space (SAVE. or GETSEG).
3Ø	ERFCU%	Cannot update file.
31	ERLOH%	Low segment overlaps high segment (GETSEG or RUN), or page overlap error (MERGE.).
32	ERNLI%	The user is not logged in (RUN, SAVE. only).
33	ERENQ%	The file has outstanding locks set.
34	ERBED%	The file has a bad EXE file directory (GETSEG, RUN, MERGE.).
35	ERBEE%	The file has a bad extension for an .EXE file (GETSEG, RUN, MERGE.).
36	ERDTB%	The file's EXE directory is too big (GETSEG, RUN, MERGE.).
37	ERENC%	The network capacity has been exceeded; not enough space for the connect message (LOOKUP, ENTER).
4Ø	ERTNA%	The task was not available (LOOKUP, ENTER, RENAME).
41	ERUNN%	An unknown network node was specified, or the node went down during the connect (ENTER).
42	ERSIU%	SFD is in use (RENAME).
43	ERNDR%	File has an NDR (No Delete or Rename) lock (FILOP.).
44	ERJCH%	Job count high (too many simultaneous accesses).
45	ERSSL%	Cannot rename SFD to lower level.
46	ERCNO%	Channel not OPENed (FILOP.).
47	ERDDU%	Device is not useable; it is offline.
5Ø	ERDRS%	Device is restricted.
51	ERDCM%	Device is under control of Mountable Device Allocator (MDA) (GALAXY).
52	ERDAJ%	Device is allocated to another job.
53	ERIDM%	Illegal data mode specified (FILOP.).
54	ERUOB%	Unknown or undefined bits set (OPEN).
55	ERDUM%	Device is in use on an MPX-controlled channel.
56	ERNPC%	No per-process space available for extended I/O channel.
57	ERNFC%	No free channels are available.
6Ø	ERUFF%	Unknown FILOP. function.
61	ERCTB%	Channel too big.
62	ERCIF%	Function illegal on this channel.
63	ERACR%	Address check occurred while reading arguments.
64	ERACS%	Address check occurred while storing arguments.
65	ERNZA%	A negative or zero argument count was specified.
66	ERATS%	Argument block was too short.
67	ERLBL%	Magnetic tape labelling error.

CHAPTER 12

DISKS (DSK)

Disks store ordered sets of data called files. One or more disk units can be associated by the monitor as file structures. The physical unit of the disk is generally transparent to users; therefore you do not need a detailed knowledge of disk units to use disks effectively.

Your system may have several types of disks. The monitor's disk-service routines handle all disk operations, including file structuring, executing disk-specific monitor calls, queueing disk requests, and optimizing disk usage.

The monitor handles all disk file structures as logical units first, then converts these to physical units in its device-dependent service routines. All disk addresses discussed in this manual are logical, or relative, addresses, not physical locations on the disk.

The basic unit on the disk is the logical disk block, which has 200 octal (128 decimal) 36-bit words of storage. A disk file can be any length, and you can store as many disk files as your disk quota will allow.

12.1 DISK NAMES

Each disk file structure has a SIXBIT name; this name is defined by the operator at system initialization time. A file structure name can be up to four alphanumeric characters in length, but must not duplicate any device name, unit name, existing file structure name, or ersatz device name.

Public file structures names should be of the form DSKn, where **n** is A to Z. Public file structures are created by the system administrator at ONCE-only time.

When your program issues an OPEN, INIT, or FILOP. monitor call, it specifies a file structure. For subsequent LOOKUP or ENTER calls, the monitor searches only the file structure named in the OPEN, INIT, or FILOP. call that initialized the channel. Therefore, to initialize a file structure, you must know the name of the file structure that contains the required file.

Often a program does not initialize a file structure, but instead initializes the generic disk device name DSK. The monitor then searches the user's job search list to determine which file structure to use. See Section 12.8 for a discussion of job search lists.

12.1.1 Logical Unit Names

If your program specifies a single file structure name (such as DSKA), it is implicitly referring to all units in that file structure. However, your program can specify a logical unit within a file structure in the form DSKnm. n is an alphabetic character representing structure; m is a unit number.

When your program reads a file, the monitor generalizes a logical unit specification (such as DSKAØ) to the larger file structure specification (such as DSKA), which may contain more than one logical unit (such as DSKAØ and DSKA1). Therefore the monitor will locate the required file regardless of which logical unit it is on.

When your program writes a file, the monitor places the file on the logical unit specified if space is available; if space is not available, the monitor places the file on another logical unit in the same file structure. For example, if you specify DSKA1 for a file, the monitor places the file on DSKA1 if there is room; if not, it places it on another logical unit (such as DSKAØ) in the same file structure (DSKA).

Nevertheless, it can be worthwhile to specify logical units for files, because file processing usually proceeds faster if the files are on different logical units.

12.1.2 Physical Controller and Disk Unit Names

Your program can refer to a disk by the generic name DSK, by a file structure name (such as DSKA), by a logical unit name (such as DSKAØ), by a controller class name, by a controller designation, or by a physical disk unit name.

Controller classes, physical controller names, and the names of the physical disk units they control are:

- Controller class RP. Physical controller names for this class are of the form RPc, where c is A-Z.

  Physical disk unit names for this class are of the form RPcn, where c completes the controller name; and n is the disk unit number (in the range Ø to 7). RP designates RH2Ø controllers with RPØ4, RPØ6, RPØ7 units, or RH11 controllers (KS1Ø only) with RPØ6 or RMØ3 units.

- Controller class RN for RP2Ø disk devices on a DX2Ø/RH2Ø controller. Physical controller names for this class are of the form RNc, where c is A-Z.

  Physical disk unit names for this class are of the form RNcn, where c completes the controller name; and n is the disk unit number (in the range Ø to 15).

- Controller class RA for an RA6Ø, RA8Ø, or RA81 drive on an HSC-5Ø controller. Physical controller names for this class are of the form RAc, where c is A-Z.

  Physical disk unit names for this class are of the form RAcn, where c completes the controller name; and n is the disk unit number (in the range Ø to 255).

12.1.3 Abbreviations

You can abbreviate disk names in ASSIGN commands and OPEN, INIT, FILOP., LOOKUP, and ENTER monitor calls. In creating files, the monitor places the file on the first disk that begins with the abbreviation you used. In searching for files, the monitor searches all disk units that begin with the abbreviation until it finds the file. The LOOKUP/ENTER calls apply to as wide a class of units as possible. For example, MO might include MONI, MONZ, and MOBY.

12.2 DISK FILE NAMES

A disk file has a file name and an extension. The file name is a SIXBIT string of up to 6 characters. The extension is a SIXBIT string of up to 3 characters.

Most programs that scan file names accept them in the format **filnam.ext**, where **filnam** is the file name, and **ext** is the extension. The file name cannot be nulls , but the extension can. Most programs that scan file names will interpret a file as having a null extension if it is written with a period after the file name. For example, the system interprets:

```
MYFILE.
```

as the file name and a null extension.

In monitor calls, you use SIXBIT to specify the file name and extension. For example, you specify the file TEST.TST in a monitor call as:

```
SIXBIT/TEST/
SIXBIT/TST/
```

When you create a file, a file name is associated with the file. The name remains associated with the file until you delete or rename the file.

12.3 DISK FILE PROTECTIONS

Every disk file has a protection code that indicates the users who can and cannot access the file. The protection code in a file specification appears as:

```
<xyz>
```

Where the first digit (x) refers to the owner field, the second digit (y) refers to users with the same project number as the owner, and the third digit (z) refers to all other users.

NOTE

Directory files (*.UFD and *.SFD) are protected by codes that appear similar to data file protection codes, but the meaning of the codes is different. For information about directory files, refer to Section 12.6.

The protection code for a file is stored in its RIB, and contains three 3-bit fields:

1. The first 3-bit field gives a protection code that determines access by the owner of the file.

2. The middle 3-bit field gives a protection code that determines access by users with the same project number as the owner of the file.

3. The last 3-bit field gives a protection code that determines access for all other users.

When the monitor symbol INDPPN=Ø (default), the owner of a file is the user whose project and programmer number matches the User File Directory containing the file. The actual definition of a file owner is set at monitor generation time using the MONGEN program. (If the symbol INDPPN is set to <Ø,,-1> with MONGEN, the owner of the file is any user whose programmer number matches the UFD containing the file.) No matter who an installation defines as a file owner, project numbers less than 1Ø are always independent of programmer numbers. For example, a user having a PPN of [1234,4] is not considered the owner of files in [1,4]. The setting of INDPPN can be obtained from GETTAB table %CNSTS, bit ST%IND.

The file owner may be protected from inadvertently destroying his files by the access protection indicated by the first field in the protection code <nnn>. The owner protection code also specifies whether the File Daemon (FILDAE) should be called on attempts to access a file. Specifically, the digit in the owner field means the following:

Code	Meaning
Ø	Any access is allowed. The owner can execute, read, append to, update, write, rename, and change the protection of his file.
1	Same as code Ø.
2	Any access to the file, except for renaming it, is allowed.
3	The owner cannot append to, update, or write to the file. The owner can execute, read, rename, or change the protection code of the file.
4	This code is equivalent to Ø and 1. However, if the File Daemon is running, any attempt to access this file that results in a protection failure will result in a call to the File Daemon. (Refer to Section 12.4.)
5	This code is equivalent to code 2, but the File Daemon is called on any attempted access that violates the protection codes.
6	The owner cannot append to, update, rename, or write to the file. The owner can execute, read, or change the protection code of the file. The File Daemon is called on any attempted access that violates the protection code.

Code	Meaning
7	The owner can execute, read, and change the protection of the file. The File Daemon is called on any attempted access that violates the protection code.

Note that codes 4 through 7 specify that the File Daemon program should be called when any users (owner or others) attempt to access the file in a manner that results in a violation of the protection code. The function of the File Daemon is discussed in Section 12.4. Table 12-1 illustrates the access allowed to the file owner for each code. Capabilities are indicated by Y, prevention against that type of access is indicated by N.

Table 12-1: File Access Protection -- Owner Field

Access Type	Code							
	Ø	1	2	3	4	5	6	7
EXECUTE	Y	Y	Y	Y	Y	Y	Y	Y
READ	Y	Y	Y	Y	Y	Y	Y	Y
APPEND TO	Y	Y	Y	N	Y	Y	N	N
UPDATE	Y	Y	Y	N	Y	Y	N	N
WRITE	Y	Y	Y	N	Y	Y	N	N
RENAME	Y	Y	N	Y	Y	N	N	N
CHANGE PROTECTION	Y	Y	Y	Y	Y	Y	Y	Y
CALL FILDAE	N	N	N	N	Y	Y	Y	Y

The second field of the protection code <nnn> applies to users in the same project group (that is, having the same project number) as the owner. The third field applies to all other PPNs. The codes for these fields have the same meaning to be applied to the different types of users. The meanings of the codes are:

Code	Symbol	Meaning
Ø	.PTCPR	The user is allowed any access to the file. He can execute, read, append to, update, write, rename, and change the protection code for the file.
1	.PTREN	The user is not allowed to change the protection of the file. However, the user can execute, read, append to, update, write, or rename the file.

Code	Symbol	Meaning
2	.PTWRI	The user is not allowed to rename or change the protection of the file. However, the user can execute, read, append to, update, or write the file.
3	.PTUPD	The user cannot write, rename, or change the protection. However, the user can execute, read, append to, or update the file.
4	.PTAPP	The user cannot update, write, rename, or change the protection of the file. However, the user can execute, read, or append to the file.
5	.PTRED	The user cannot append to, update, write, rename, or change the protection of the file. However, the user can execute or read the file.
6	.PTEXO	The user can only execute the file.
7	.PTNON	The user cannot access the file.

Table 12-2 shows the access allowed and prevented by each code for each type of access by project members and by other users.

Table 12-2: File Access Protection -- Second and Third Digits

Access Type	Code							
	Ø	1	2	3	4	5	6	7
EXECUTE	Y	Y	Y	Y	Y	Y	Y	N
READ	Y	Y	Y	Y	Y	Y	N	N
APPEND TO	Y	Y	Y	Y	Y	N	N	N
UPDATE	Y	Y	Y	Y	N	N	N	N
WRITE	Y	Y	Y	N	N	N	N	N
RENAME	Y	Y	N	N	N	N	N	N
CHANGE PROTECTION	Y	N	N	N	N	N	N	N

The greatest protection that a file can have is code 7, and the least protection is code Ø. Usually, the owner's field is Ø or 1. It is always possible for the owner of a file to change the protection code associated with his file, even if the owner's protection code is set to 7. Therefore, codes Ø and 1 are equal when they appear in the owner's field.

You can change the file access protection code by issuing the RENAME monitor call, the FILOP. monitor call with the RENAME option, or the PROTECT command.

When your program issues an ENTER that does not specify a protection code and the file does not exist, the monitor substitutes either:

- The standard protection code.
- The default protection code that you specified either by the SET DEFAULT PROTECTION command or by the SETUUO monitor call.

The normal system standard protection code is Ø57. This protection code prevents users in different projects from accessing another user's files; however, a standard protection of Ø55 is recommended for systems where privacy is not as important as the capability of sharing files among projects.

In fact, the monitor automatically assigns default protection codes to system files that it must be able to access. For SYS:*.SYS files (those files in directory [1,4] with file extension .SYS) the default protection code is <157>. All other files on SYS are assigned protection code <155>.

Default protection codes are defined by symbols that can be changed using the HDWGEN portion of MONGEN.

However, no program should be coded to assume knowledge of the standard protection code; it should be obtained through a GETTAB monitor call. The relevant GETTAB items are:

%LDSTP - Standard file protection.
%LDUFP - Standard directory protection.
%LDSPP - Spooled file protection.
%LDSYP - Standard SYS protection.
%LDSSP - SYS:*.SYS protection.

You can set a default file protection code by using either the SET DEFAULT PROTECTION command or the .STDEF function of the SETUUO monitor call. If you (or your program) set a default protection code by either of the above methods, the monitor creates the file with the default code if the ENTER block contains zero in its file protection code field. If you (or your program) did not specify a default protection code or specified the SET DEFAULT PROTECTION OFF command, the monitor creates the file with the installation's default file protection code. The SET DEFAULT PROTECTION OFF/ON command turns off/on the previous setting of the default protection code. (Refer to the TOPS-1Ø Operating System Commands Manual for more information on the SET DEFAULT PROTECTION command.) Programs that set a default protection code should GETTAB the installation's default protection code (unless FILDAE has specified otherwise) and then set the user default protection code to that returned on the GETTAB, if the default protection code is desired.

12.4 THE FILE DAEMON (FILDAE)

A File Daemon is a privileged program that serves the following functions:

- o Oversees file accessing.
- o Aids in accounting.
- o Tracks program use.

DIGITAL provides and supports the interface to a File Daemon. DIGITAL provides but does not support a File Daemon (FILDAE); each installation should write and support its own File Daemon to meet its own requirements.

When a File Daemon is running, the monitor calls it every time someone tries to access a file that has a 4, 5, 6, or 7 in the owner's protection code field and the access fails due to a protection error. The monitor also calls a File Daemon for a directory when attempted access fails due to a protection error. (Appendix C contains a description of the File Daemon and the ACCESS.USR file.)

12.5 DISK FILE FORMATS

A disk file contains the data that makes up the file, and information that the monitor needs to retrieve the file. Each disk block is 200 (octal) words long.

The monitor writes a full block on each disk write (when your program outputs a buffer). Any unused portion of the block is filled with zeros, but the monitor keeps track of the actual length of the last block in the file only. Therefore if your program outputs a buffer that is not full, the block is filled with zeros; but on reading the block, it will appear to be a full block of data.

The first data block for a file is pointed to by an entry in the retrieval information block (RIB) for the file. The RIB is pointed to by an entry in a UFD or an SFD. Thus there is a chain from the directory through the RIB to the first block of the file.

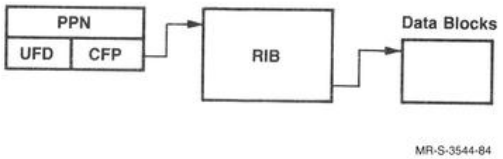


Figure 12-1: Disk Chain

The RIB for a file contains pointers to the entire file. At the physical end of a file (unless it is open), there is a copy of the RIB for the file. This spare RIB is the block immediately following the last data block of the file. Users are not allowed to access the spare RIB. Thus the file has two overhead RIB blocks: one for the prime RIB (in relative block 0) and one for the spare RIB (in the last relative block of the file).

12.6 DISK DIRECTORIES

A directory is itself a file; it serves as an index to other files on a device. You can read a directory like any other file, but you cannot write a directory. Each file structure has directories arranged in a tree structure of three levels:

1. The Master File Directory (MFD) is at the root of the tree. It serves as an index to User File Directories (UFDs) on the file structure.

2. The User File Directories (UFDs) are at the next level of the tree structure. UFDs contain pointers to and data about user files and subfile directories (SFDs).

3. Subfile Directories (SFDs) are at the remaining levels of the tree structure. SFDs contain pointers to and data about user files and subordinate SFDs.

The general disk file organization for a file structure is shown in Figure 12-2. Figure 12-3 shows a more detailed disk file organization.

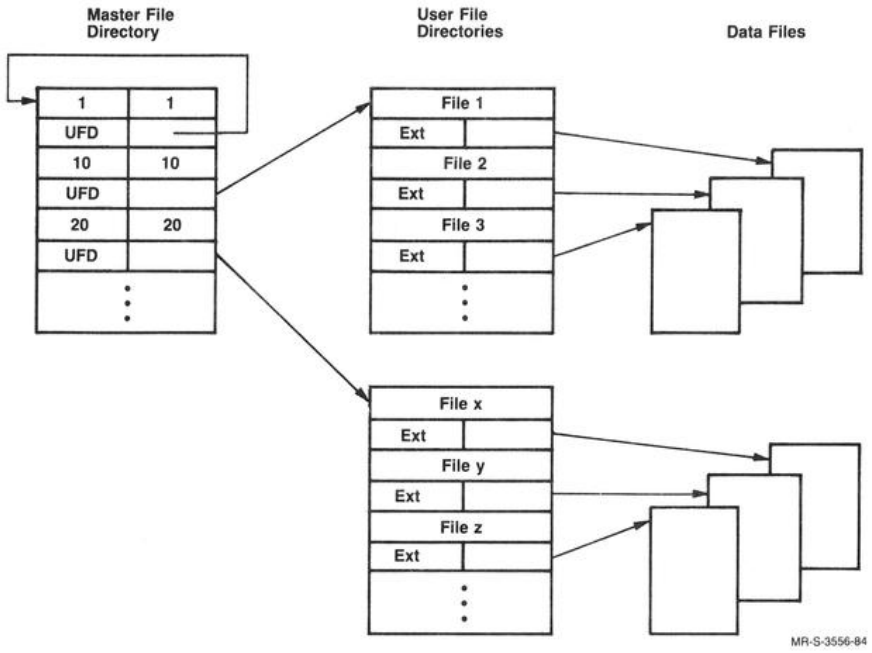


Figure 12-2: General Disk File Organization for a File Structure

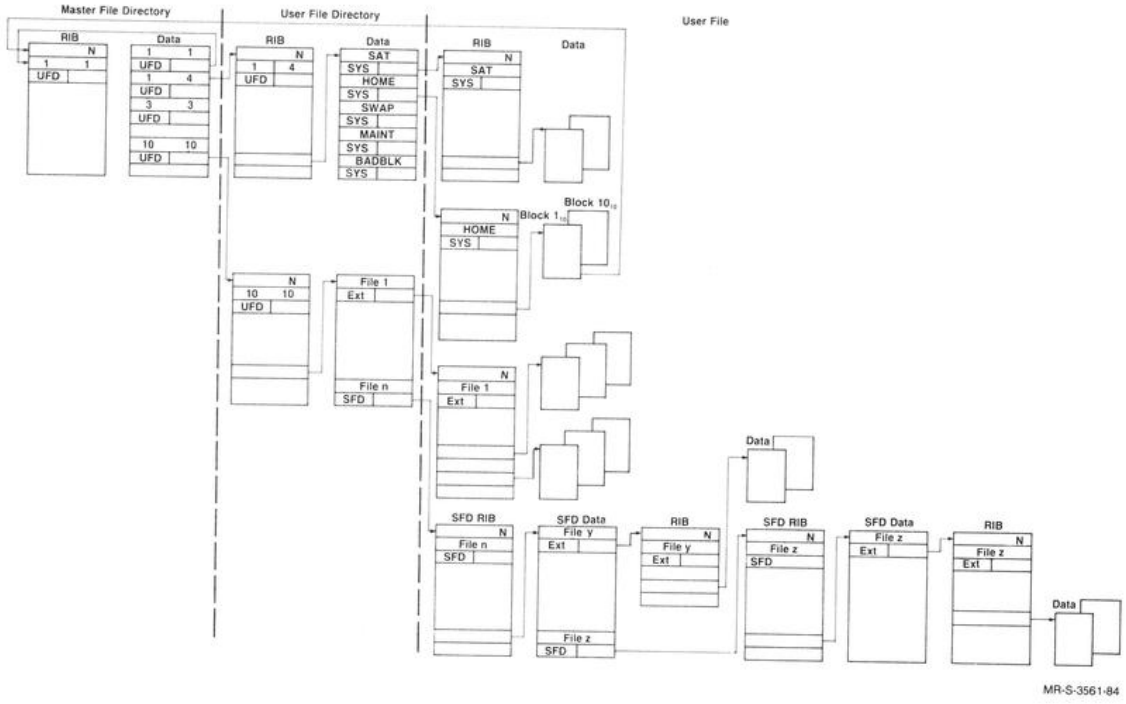


Figure 12-3: Disk File Organization

12.6.1 The Master File Directory (MFD)

The Master File Directory is a directory of all the individual user file directories on the file structure. It consists of 2-word entries. Each entry gives the name and address of a user file directory (UFD) in the file structure.

Each entry in the MFD is in the format:

```
XWD     proj,prog           ;Project-programmer number
XWD     'UFD',CFP           ;UFD,,compressed file pointer to UFD
```

where proj and prog give the project-programmer number (PPN) for a UFD. UFD is the name of the UFD in SIXBIT. The Compressed File Pointer (CFP) is the supercluster number of the RIB for the UFD.

The MFD contains an entry for each UFD on the system. A "continued MFD" is the set of all MFDs for all file structures in a job's search list.

12.6.2 User File Directories (UFDs)

A User File Directory (UFD) is a list of the names of files existing in a given project-programmer area within the file structure. It consists of 2-word entries. Each entry gives the name and address of a user file, or a Subfile Directory (SFD).

Each entry in the UFD is in the format:

```
SIXBIT  /filename/          ;File name
XWD     'extension',CFP     ;Extension,,CFP to file
```

where: **file name** is a SIXBIT file name of up to six characters; **extension** is a SIXBIT file extension of up to three characters; and CFP is the compressed file pointer to the RIB of the file. A continued UFD is the set of all UFDs on all file structures in the job's search list for the given PPN.

When you log in, each file structure on which you have disk space contains a UFD for your PPN. Each UFD indexes your files for that file structure only.

UFDs are created by programs, such as LOGIN, CREDIR, and PULSAR when the program is run in a privileged account, such as [1,2]. Only privileged programs can create UFDs, and only the monitor can write UFD data.

Any program can attempt to read a UFD. Whether the attempt succeeds depends on the directory protection code for the UFD. See Section 12.7 for a discussion of directory protections.

12.6.3 Subfile Directories (SFDs)

A Subfile Directory (SFD) consists of 2-word entries in the same format as a UFD entry. The UFD or a superior SFD points to an SFD; each SFD entry points to a user file or to a subordinate SFD. Unlike UFDs, SFDs can be created by any program. Figure 12-3 illustrates file organization.

The maximum number of levels for SFDs (which cannot be more than five) is a MONGEN parameter; you can obtain this value from the right half of the item %LDSFD in the GETTAB Table .GTLVD. A "continued SFD" is the set of all SFDs on all file structures in a job's search list that have the same PPN and directory path. See Section 12.6 for a discussion of directory paths.

SFDs can be useful to your programs because they allow you to organize files in your area; for example, you might group files according to their functions. Files that have the same name, but are in different SFDs, are uniquely identifiable. Therefore simultaneous batch runs of the same program for a single user can use the same file names without conflicting with each other.

You can create subfile directories by using the CREDIR program, which is described in the TOPS-10 User Utilities Manual. You can delete SFDs using either the DELETE command or the delete function of the RENAME call. Note, however, that an error will occur if you try to delete an SFD when it is included in your job's default path in your job search list, or when it contains files.

12.6.4 Directory Paths

A disk file is uniquely identifiable by a string giving its file structure name, its directory path, and its file name and extension. The directory path is an ordered list of directory names (without regard to file structure). This path always begins with a UFD.

Your program can use the PATH. monitor call to read or set the default directory path for your job. A default path can contain any of the following:

- Your job's UFD.
- Your job's UFD and one or more SFDs in a chain originating in your UFD.
- A UFD different from your job's UFD.
- A UFD different from your job's UFD and one or more SFDs in a chain originating in that UFD.

The initial default path for a job is your UFD. To specify a different path, you must give the path in the format:

```
[projno,progno,sfd1,sfd2,...]
```

where projno and progno give the project-programmer number, or the UFD. sfd1 is the name of a subfile directory pointed to by a file in the UFD; sfd2 is the name of a subfile directory pointed to by a file in sfd1, and so forth.

For example, if you have not changed the initial default path for your job, then the commands

```
.R MACRO
*FOO.REL=FOO.MAC
```

specify the files FOO.REL and FOO.MAC in your job's UFD (that is, your PPN).

However, you can specify a different path. The commands:

```
.R MACRO
*FOO.REL=MINE:FOO.MAC[27,5031,OLD]
```

specify FOO.REL in your UFD and FOO.MAC in the SFD called OLD in the UFD for PPN [27,5031] on the file structure MINE:.

12.6.5 Pathological Device Names

TOPS-10 allows logical names for disk directory paths, as well as specific devices. The logical name for a device can be obtained using the DEVNAM monitor call. A logical name for a directory path is called a "pathological name." You can obtain the pathological name for a device using the PATH. monitor call, .PTFRN function. You can set pathological names using the DEVLNM monitor call, or the .PTFSN function of the PATH. monitor call.

You can use the PATH. monitor call to define, read, and delete logical path names. You could define the pathological name to be a path including multiple file structures, UFDs, and SFDs. For example, to define FOO: to be the pathological name for the following:

```
DSKB:[10,664,A],ALL:[10,675],NEW:[27,4072]
```

you could issue the following monitor call:

Example

```
                MOVE    AC1,[XWD ARGLEN,ARGLST]
                PATH.   AC1,
                   JRST ERR
                JRST    NORM

ARGLST:         EXP .PTFSN
                EXP Ø
                SIXBIT /FOO/
                EXP Ø
                SIXBIT /DSKB/
                EXP Ø
                EXP Ø
                XWD 1Ø,664
                SIXBIT /A/
                EXP Ø
                EXP Ø
                SIXBIT /ALL/
                EXP Ø
                EXP Ø
                XWD 1Ø,675
                EXP Ø
                EXP Ø
                SIXBIT /NEW/
                EXP Ø
                EXP Ø
                XWD 27,4Ø72
                EXP Ø
                EXP Ø
                EXP Ø
                ARGLEN = .-ARGLST
```

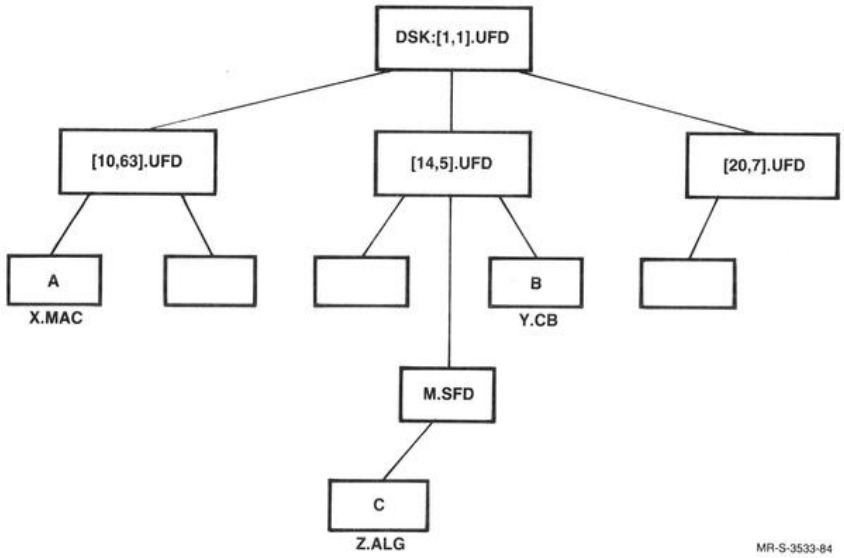


Figure 12-4: Directory Paths on a Single File Structure

Refer to Figure 12-4. The path for File A is:

```
X.MAC[10,63]
```

The path for File B is:

```
Z.ALG[14,5,M]
```

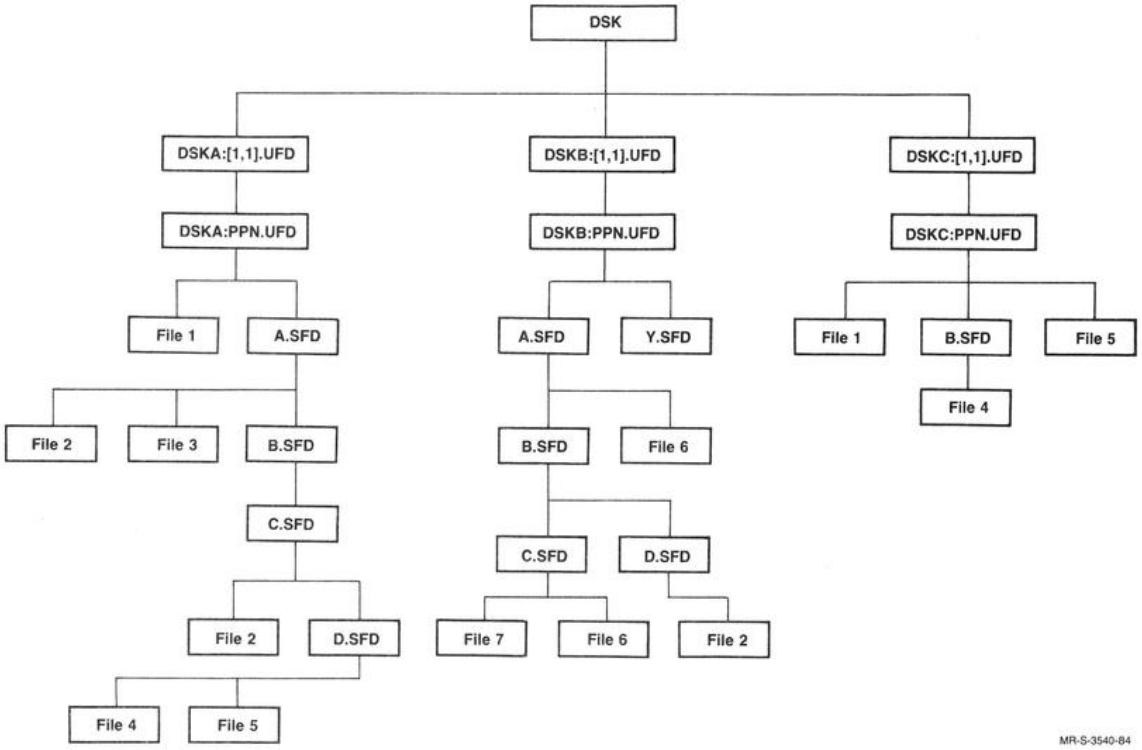


Figure 12-5: Directory Paths on Multiple File Structures

Refer to Figure 12-5. The job's search list in this figure is:

```
DSKA,DSKB,DSKC
```

The job's default path is:

```
[PPN,A,B,C]/SCAN
```

The following describes the order of the monitor's search when the job issues LOOKUP and ENTER monitor calls. The order of the search for a file with /SCAN set by SETSRC, is:

1. DSKA:[PPN,A,B,C]
2. DSKB:[PPN,A,B,C]
3. DSKC:[PPN,A,B,C]
4. DSKA:[PPN,A,B]

5. DSKB:[PPN,A,B]
6. DSKC:[PPN,A,B]
7. DSKA:[PPN,A]
8. DSKB:[PPN,A]
9. DSKC:[PPN,A]
10. DSKA:[PPN]
11. DSKB:[PPN]
12. DSKC:[PPN]

If any of these SFDs do not exist, the search step involving it is omitted. Therefore, if DSKC:A.SFD[PPN] does not exist, Steps 3, 6, and 9 are omitted. If /SCAN is not set, Steps 4 through 12 are omitted. If the file exists in multiple directories, the search ends with the first occurrence found.

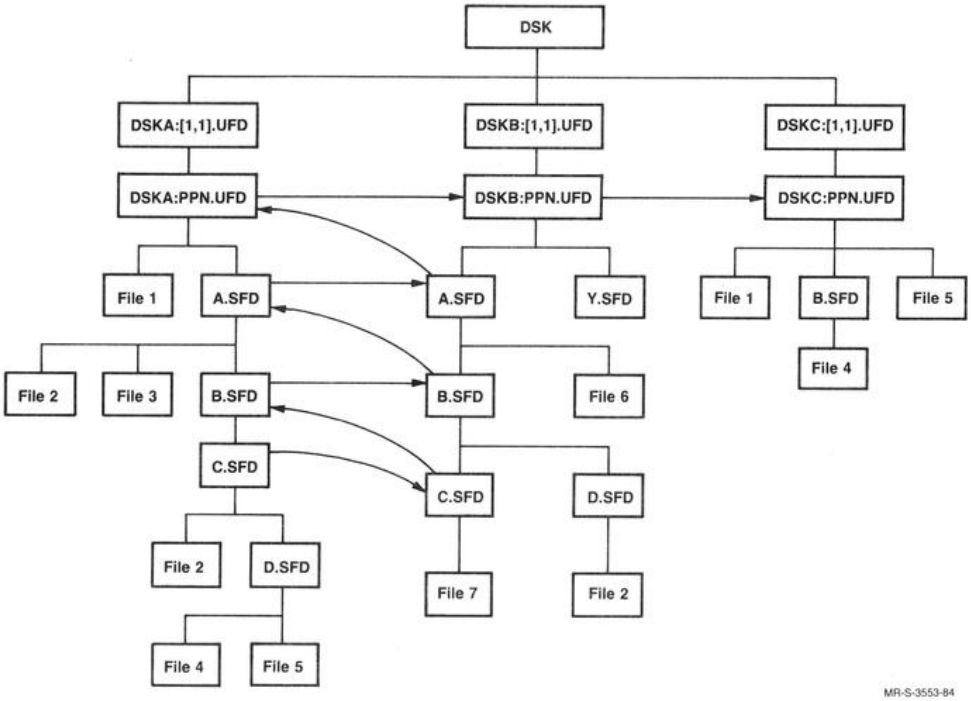


Figure 12-6: LOOKUP on DSK with No Matches

Refer to Figure 12-6. LOOKUP on SIXBIT/DSK/ with no matches results in the search that is indicated.

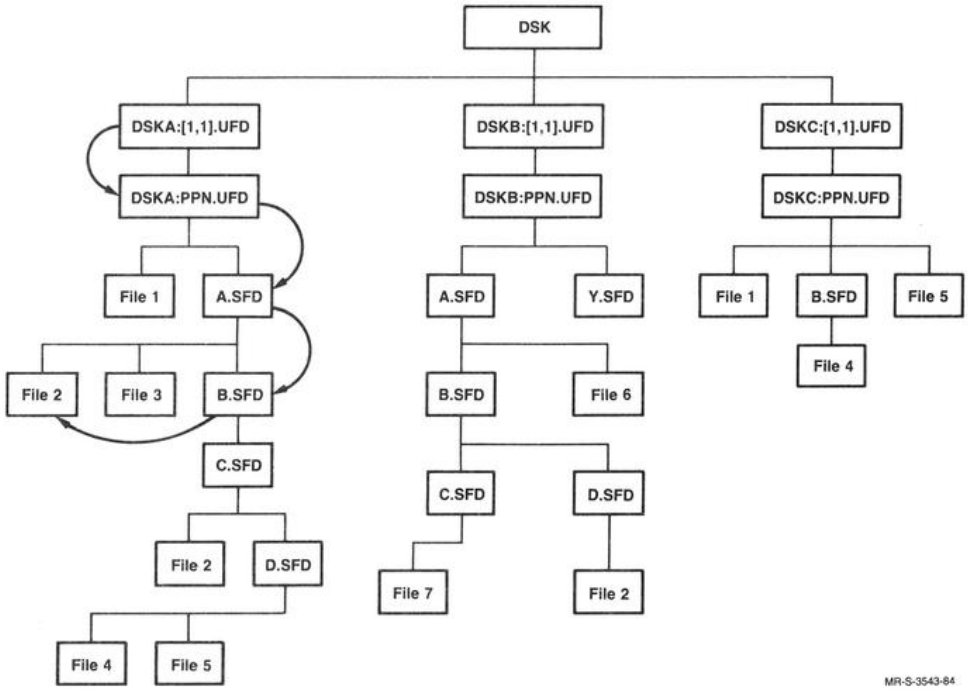


Figure 12-7: LOOKUP on DSK for FILE2

Refer to Figure 12-7. LOOKUP on SIXBIT/DSK/ and SIXBIT/FILE2/ results in DSKA:FILE2[PPN,A,B].

LOOKUP on SIXBIT/DSKB/ and SIXBIT/FILE2/ or LOOKUP on SIXBIT/DSKC/ and SIXBIT/FILE2/ fails.

ENTER on SIXBIT/DSK/ and SIXBIT/FILE9/ results in an error because no file structure has both the no-create bit cleared and the directory structure [PPN,A,B,C,D].

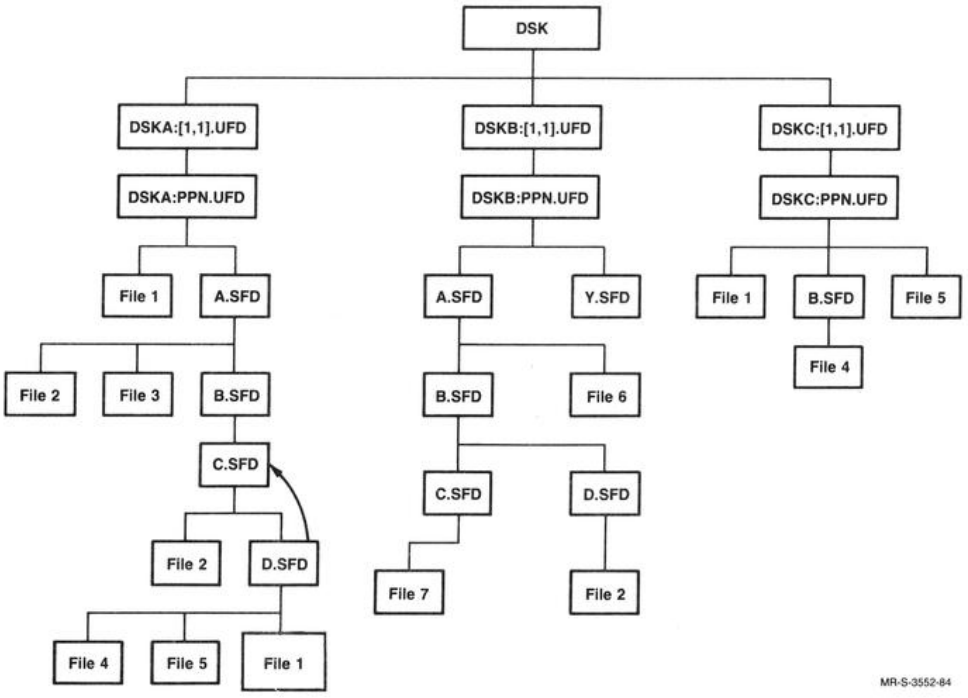


Figure 12-8: ENTER on DSKA for FILE1

Refer to Figure 12-8. ENTER on SIXBIT/DSKA/ and SIXBIT/FILE1/ results in the file being created at the end of the path on DSKA.

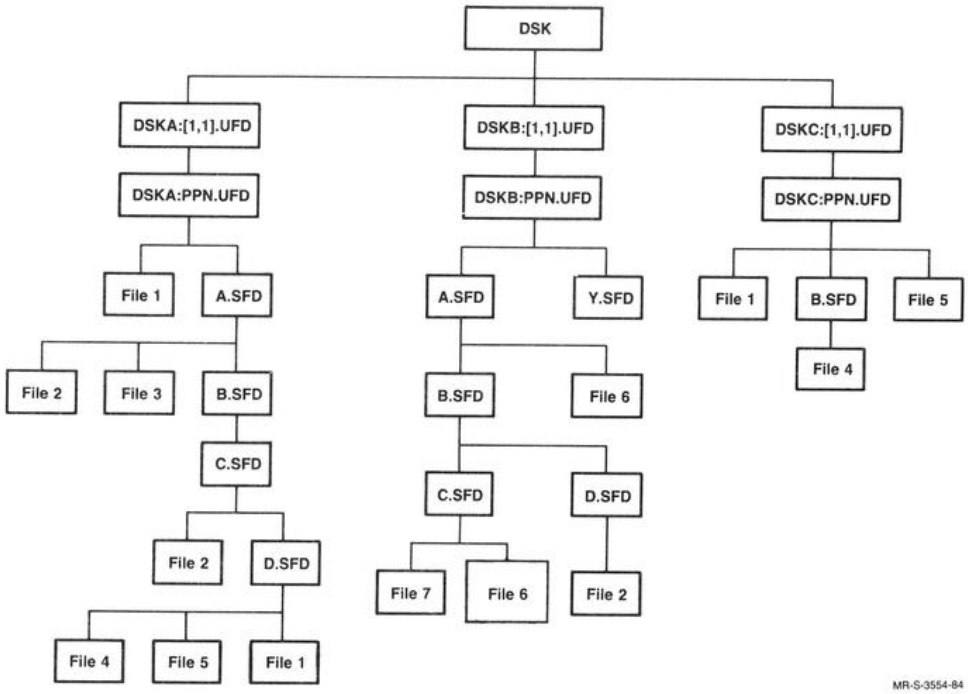


Figure 12-9: ENTER on DSK for FILE6

Refer to Figure 12-9. When the job's default path is DSKB:[PPN,A,B,C], the following calls results in the described fashion.

ENTER on SIXBIT/DSK/ and SIXBIT/FILE6/ results in the file being created on DSKB.

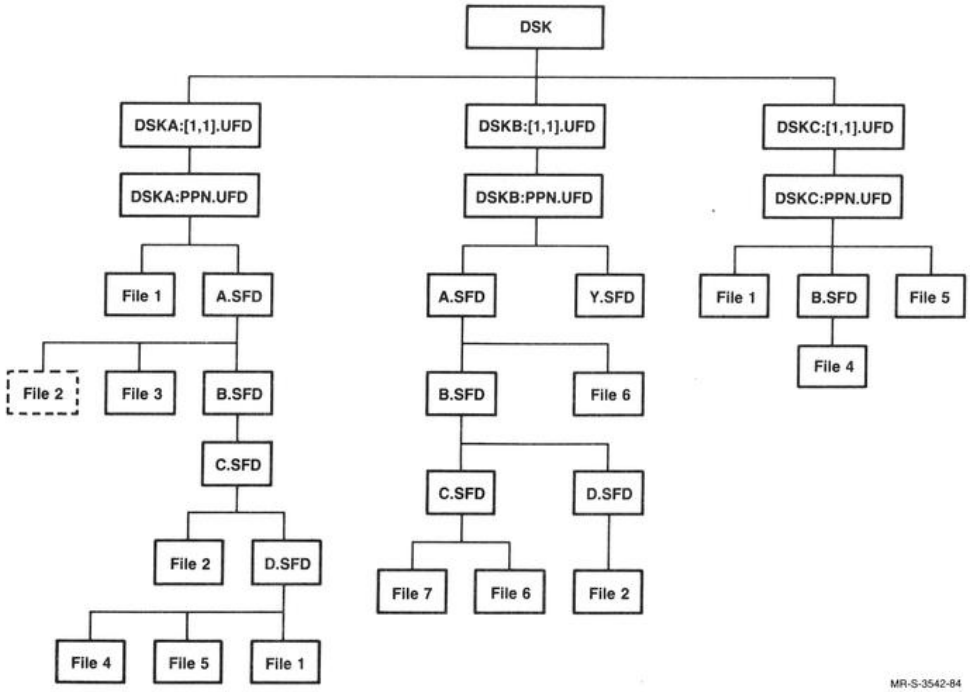


Figure 12-10: ENTER on DSK for FILE2

ENTER on SIXBIT/DSK/ and SIXBIT/FILE2/ supersedes FILE2 on DSKA:FILE2[PPN,A] as shown in Figure 12-10.

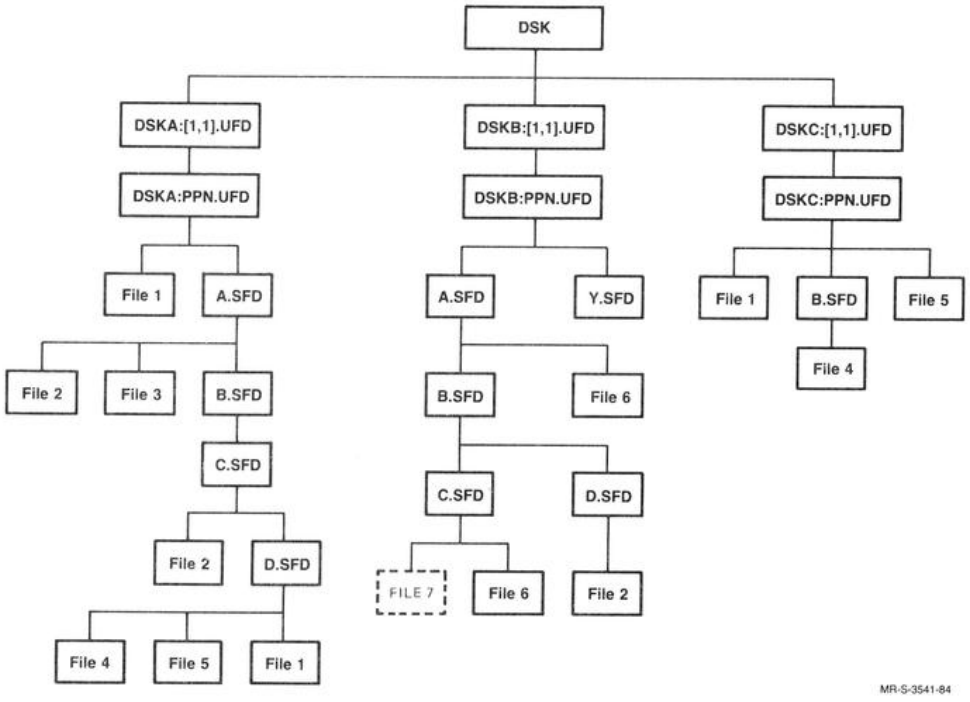


Figure 12-11: ENTER on DSK for FILE7

ENTER on SIXBIT/DSK/ and SIXBIT/FILE7/ supersedes FILE7 on DSKB, as illustrated in Figure 12-11.

Example

Assume that you set up the following path:

```
      MOVE      AC1,[XWD 5,A]
      PATH.     AC1,
        HALT    .
      MOVE      AC1,[XWD 3,B]
      PATH.     AC1,
        HALT    .

A:    .PTFSD    ;Define default path
      .PTSCY    ;Scanning in effect
      10,,63    ;UFD=[10,63]
      SIXBIT/NAME/    ;SFD=NAME
      0         ;Default path is 10,63,NAME

B:   .PTFSL     ;Define additional path
      PT.SNW + PT.SSY    ;NEW: and SYS:
      10,,7     ;User library
```

If you then logged-in as a [10,10] job and performed a LOOKUP on DSK:FILTST.EXT, the following directories would be searched:

```
[10,63,NAME]
[10,63]         ;Your search list
[10,7]

[1,5]
[1,4]           ;System search list
```

If you are logged-in as a [10,10] job and your program performs a LOOKUP on DSK:PRJFIL.EXT[10,155], the following directories are searched:

```
[10,155]
[10,7]          ;Your search list

[1,5]
[1,4]           ;System search list
```

12.7 DISK DIRECTORY PROTECTIONS

The ability of a program to create files in a directory is controlled by the protection code for the directory. The directory can be a UFD or an SFD.

By changing the protection code associated with a directory you can also specify a class of users who cannot LOOKUP any file in the directory. This specification is independent of the specification of individual files' protection codes.

Directories may be read in the same manner as any other file. The right to read a directory as a file is also controlled by the directory's protection code. Note that directory files cannot be explicitly written; therefore, the only operations possible are the following:

- LOOKUPs for files in the directory (PT.LOK).
- Creates (ENTERs and RENAMEs) for files in the directory (PT.CRE).
- Reads for the directory itself (PT.SRC).

Users are divided into the same three privilege classes for UFDs and SFDs as they are for files. Each privilege class has three independent bits, specifying a protection code. The directory protection codes are:

Code	Symbol	Meaning
0		No access allowed.
1	PT.SRC	Search directory.
2	PT.CRE	Allow file creation.
3	PT.SRC+PT.CRE	Search directory and allow creates.
4	PT.LOK	Allow LOOKUPs.
5	PT.SRC+PT.LOK	Search directory and allow LOOKUPs.
6	PT.CRE+PT.LOK	Allow creates and LOOKUPs.
7	PT.SRC+PT.CRE+PT.LOK	Search directory and allow creates and LOOKUPs.

Note that, for disk files, 7 is the highest protection code. For directories, Ø is the highest protection.

As the owner of your UFD, you can control the access to your UFD and SFDs. You can always change the protection code of your directories through the use of the RENAME monitor call. Any program can create or delete SFDs; however only privileged programs can create or delete UFDs. The monitor checks for the following types of privileged programs:

- Jobs logged in under [1,2].
- Jobs running with the JACCT bit set in JBTSTS.

Programs meeting the above requirements can do the following:

- Create UFDs and/or SFDs.
- Delete empty UFDs and/or SFDs.
- Set privileged arguments to LOOKUP, ENTER, and RENAME monitor calls.
- Ignore file protection codes unless FTFDAE = 1 and the File Daemon is running.

Privileges are similar for both UFDs and SFDs except that SFDs can be created, renamed, and deleted by both a privileged program and the owner of the SFD.

The MONGEN parameter M.XFFA can be set to prevent [1,2] jobs from being able to access files. If M.XFFA equals Ø (the default condition), FILDAE can prevent files from being accessed by [1,2]. If M.XFFA is set to -1, FILDAE is not invoked on attempted access by [1,2] jobs.

As when accessing existing files, it is possible to control the operations in a directory by using a File Daemon. Whenever an operation fails due to an access protection failure, the monitor calls a File Daemon (if one is running). Therefore, a protection code of 775 allows users in the same project to create files in the directory without the File Daemon being called. However, because of the 5 code in the other users' field, any other user's attempt to create a file in that directory causes the monitor to call the File Daemon. The File Daemon checks the ACCESS.USR associated with the directory. The file is created if the /CREATE switch is specified in the ACCESS.USR file. The File Daemon is called automatically when a directory access protection failure occurs.

12.8 JOB SEARCH LISTS

For most program uses, a file structure name serves the same purposes as a device name. Many programs use the generic device name DSK:. This causes the monitor to search a list of file structures called the job search list.

Your job search list is in two parts: the active search list and the passive search list. The active search list is searched whenever you use the generic device name DSK. The passive search list is never searched; it is checked when your job logs out, to make sure you have not exceeded your disk quota for the structures listed there.

The SETSRC program (described in the TOPS-10 User Utilities Manual) can list and change your job search list. The program reports the list in the form:

```
strname,...strname, FENCE strname,...strname
```

where each **strname** is the name of a file structure; FENCE represents the boundary between the active and passive search lists. Structures are searched in the order they are listed, from left to right. The structures listed on the left of the FENCE are the active job search list; the structures listed on the right of the FENCE are the passive search list. Each **strname** may be followed by an optional switch that indicates how the file structure can be accessed (for example, read only).

A default search list is defined for you by the system administrator when you are given your PPN for the system. When you log in, the LOGIN program sets the default search list for your job and makes sure that you have a UFD on each structure for which you have a nonzero quota. You can modify the search list for your job by:

- Using the SETSRC program (see the TOPS-10 User Utilities Manual). This program allows you to add to or delete from the list of structures in your job search list. In addition, you can use the /NOCREATE switch to SETSRC to prevent creation of files on a structure unless you have specified that structure explicitly. There are other options for SETSRC provided through other switches.

- Using the MOUNT monitor command (see the TOPS-10 Operating System Commands Manual). This command makes a structure available to your program, and adds the structure name to the end of your active search list. When you modify your search list with MOUNT, the modifications are abolished when you log off the system. When you log back on the system, your predefined search list is used.

When you request that a structure be removed from your search list (using the SETSRC program), the structure is moved from the active to the passive search list. This is done so the structure can be quota-checked when you log off the system. The LOGOUT program requires that the number of blocks allocated to a user on each file structure be less than or equal to the logged-off quota on that file structure.

If a job's search list is:

```
DSKA:/NOCREATE,DSKB:, FENCE
```

and the user OPENs a channel for device DSKA, an ENTER on that channel will create a file on DSKA. If the channel is opened for device DSK, the ENTER will create a file on DSKB.

When your program issues a LOOKUP for device DSK, the monitor searches the file structures in the order specified by your job's search list. When your program issues an ENTER specifying a non-existent file name, the monitor places the file on the first file structure in your search list that has space and does not have the NOCREATE or NOWRITE flag set. When your program issues an ENTER specifying an existing file name on any file structure in your search list, the monitor places the file on the same structure that contains the older version of the file (and the file is superseded).

When you set the NOWRITE flag, the monitor will not allow your job to write on the structure.

When you set the NOCREATE flag, you cannot create new files on the file structure, unless you explicitly specify that file structure. For example, if you set the NOCREATE flag for DSKA and you make the following specification:

```
DSKA:FOO=
```

the monitor creates FOO on DSKA because you explicitly specified DSKA. But, when you make the following specification:

```
DSK:FOO=
```

the monitor creates FOO on the first file structure specified in your job's search list that does not have the NOCREATE or NOWRITE flag set.

12.9 DISK PRIORITIES

You can use the DISK. monitor call to set and examine the parameters associated with disk file systems. DISK. allows your program to assign a priority level for disk operations such as data transfers and head positionings. You can set these priorities either for a specific I/O channel or for all channels associated with your job. When you use disk priority operations, the disk operation request with the highest priority levelis chosen. If no priorities are set, the request most satisfying disk optimization is chosen.

12.10 DISK I/O

A number of monitor calls are useful for disk I/O. The LOOKUP, ENTER, and RENAME calls can use an extended argument list for calls to disk. The extended argument list is described in Section 11.13.

The monitor calls that are especially useful for disk I/O are:

CLOSE	Closes a disk file.
DEVPPN	Returns the PPN associated with a disk device.
DISK.	Sets or returns disk file parameters.
DSKCHR	Returns disk characteristics.
ENTER	Selects a file for output.
FILOP.	Performs many file operations, including creating, deleting, writing, reading, renaming, and superseding a file.
GOBSTR	Returns file structure names from the search list for a given job or the system search list.
JOBSTR	Returns file structure names from the search list for the current job.
LOOKUP	Selects a file for input.

PATH.	Sets or reads the user's default directory path, reads the default directory path for a device or channel, or sets or reads logical name definitions.
RENAME	Renames a file. This "renaming" can include changing the file name, extension, protection for the file, and deleting the file.
STRUUO	Modifies the search list for a job or for the system.
SUSET.	Selects a logical block on a disk unit, either with respect to file structure or to unit name.
SYSPHY	Returns the names of physical disk units.
SYSSTR	Returns the names of file structures on the system.
USETI	Selects a block for next input.
USETO	Selects a block for next output.

12.11 DISK DATA MODES

Data transfers to and from disk can be made in buffered mode or dump mode.

12.11.1 Buffered Modes

The buffer size for all buffered-mode disk transfers is 203 (octal) words: 3 header words and 200 data words. The monitor always assumes this buffer size, even if you have mistakenly used smaller or larger buffers in your program. Thus, if you use a smaller buffer, the monitor reads past the end of the buffer (for example, into the header and data of the next buffer), thus destroying that data. By using the SET BIGBUF command, you can enable larger buffers for disk I/O. With big buffers, your program transmits and receives disk I/O in multiples of 200 words. For example, the command:

```
.SET BIGBUF 3
```

will result in 603 (octal) words: a 3-word header and 3*200=600 data words.

The buffered data modes that you can use for disk transfers are:

Code	Symbol	Meaning
0	.IOASC	ASCII mode
1	.IOASL	ASCII line mode
10	.IOIMG	Image mode
13	.IOIBN	Image binary mode
14	.IOBIN	Binary mode

All of these data modes are transferred without processing by the monitor.

12.11.2 Dump Modes

In dump modes, data can be read from or written into any locations in your user memory area. For output to disk from user memory, the disk-service routine uses disk space in increments of 200 (octal) words, filling the last 200-word block with zeros if necessary.

The dump data modes that you can use for disk transfers are:

Code	Symbol	Meaning
16	.IODPR	Dump record mode.
17	.IODMP	Dump mode.

12.12 DISK I/O STATUS

I/O status bits for disk are as follows:

Bits	Symbol	Meaning
18-21	IO.ERR	Error flags:

Flag	Symbol	Error
18	IO.IMP	Attempt to write on software write-locked file structure or software redundancy failure occurred.
19	IO.DER	Error detected by device.
20	IO.DTE	Hard data error.
21	IO.BKT	Block too large.

22	IO.EOF	End of file.
23	IO.ACT	Device active.
29	IO.WHD	Write disk pack headers.
30	IO.SYN	Synchronous I/O.
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
0	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
10	.IOIMG	Image mode.
13	.IOIBN	Image binary mode.
14	.IOBIN	Binary mode.
16	.IODPR	Dump record mode.
17	.IODMP	Dump mode.

CHAPTER 13

DECTAPES (DTA)

A DECtape device reads and writes a DECtape. It is a sequential device, and has a directory. The tape is 260 feet (79.24 m) long and 10 tracks wide.

13.1 DECTAPE DEVICE NAMES

The physical name of a DECtape unit is of the form:

DTcnnu

where **c** is either A or B; **nn** is the node number of the CPU; and **u** is the unit number in the range 0 to 7.

A CPU may have up to two DECtape controllers. The name of the DECtape controller is TD10.

The unit name of the DECtape device is either TU55 or TU56.

13.2 DECTAPE DATA MODES

A DECtape device can be operated in any of the following data modes:

- Buffered modes: ASCII, ASCII line, image, image binary, and binary.
- Unbuffered data modes: dump record and dump.
- Nonstandard data mode.
- Semistandard data mode.

13.2.1 Buffered Data Modes

The buffered data modes (ASCII, ASCII line, image, image binary, and binary) use buffers of 200 (octal) data words. Data is transferred between the DECtape device and memory without change. Checksumming is performed by the DECtape service routine.

In buffered mode, the buffers are 200 (octal) words long, but data is still in blocks of 177 (octal) words (see Figure 13-1).

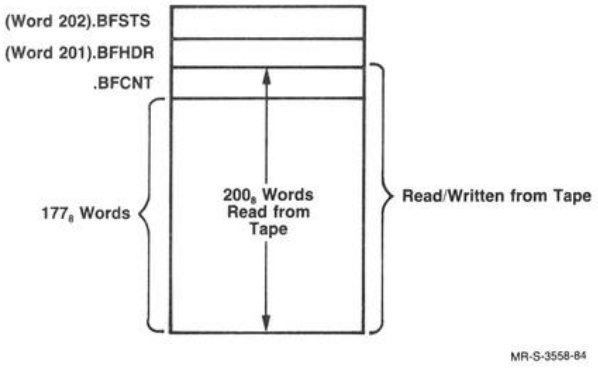


Figure 13-1: DECtape Buffer

The first word in the buffer (.BFCNT) is the link pointer directly to the tape. Ususally, the monitor passes over this word. Programs that require .BFCNT may read it using buffered I/O mode.

13.2.2 Unbuffered Data Modes

The unbuffered data modes (dump record and dump) use command lists to transfer data between DECtape and memory. The I/O monitor call (IN, INPUT, OUT, or OUTPUT) gives the address of the command list.

Each word of the command list is in one of the following forms:

```
IOWD    buflength,buffer
XWD     0,nextcmd
Z
```

where in the IOWD instruction, **buflength** is the number of words to be transferred and **buffer** is the address for reading or writing the data; in the XWD instruction, **nextcmd** is the address of the next command in the list; and the Z (zero) word ends the command list.

File-structured dump mode data is blocked into standard DECtape blocks by the DECtape service routine. Each block read or written contains a link word and 177 (octal) data words. On output, if the data does not fill the last block, it is left blank. On input, your program must read only the words that were written, skipping these blank words.

13.2.3 Nonstandard Data Mode

Your program selects nonstandard DECtape data mode by setting bit IO.NSD in the I/O status word (using the SETSTS monitor call). In nonstandard mode, the monitor does not perform file-structuring operations on the data. The monitor reads and writes each block sequentially, but it does not generate links on output, nor does it recognize links on input.

Nonstandard mode treats link words on the DECtape as data words; therefore a data block is up to 200 (octal) words, and your program must select the block to be read or written by using the USETI or USETO monitor call.

A LOOKUP, ENTER, or UTPCLR monitor call to a DECtape device in nonstandard mode is a no-op. Your program is prohibited from reading or writing Block Ø of the tape in dump mode, because this block cannot be read or written in a forward direction.

The monitor checks block numbers for validity, but does not perform dead-reckoning computations for the requested blocks (because a block may be shorter than 2ØØ words).

13.2.4 Semistandard Data Mode

Semistandard mode allows you to specify block numbers greater than 11Ø1 (octal). It is useful for DECtapes formatted on machines other than DECsystem-1Øs, for example, a PDP-8. The block size of such tapes might not be 2ØØ words; therefore, semistandard mode does not limit the tape to 578 blocks of data.

To select semistandard mode, your program must set both the IO.SSD and IO.NSD bits in the I/O status word (using the SETSTS monitor call). Semistandard mode is identical to nonstandard mode, except that the monitor:

- Checks block numbers to determine position.
- Starts the tape in the same direction it last went.
- Performs dead-reckoning computations for the requested blocks. (Refer to Section 13.3.)

13.3 DECTAPE I/O

On the first LOOKUP, ENTER, or UGETF call to a DECtape device, the DECtape service routine reads the directory into memory. The directory is maintained and updated in memory until the device channel is released by a RELEAS monitor call; at that time the directory is copied to the DECtape.

When you change DECtapes on a DECtape device, you should issue an ASSIGN command to inform the monitor that a new tape is on the device; otherwise, the wrong directory may be copied onto the tape.

If you use monitor calls (USETI and USETO) to position a DECtape, the monitor uses **dead reckoning** to locate the correct block on the tape. This means that the service routine begins turning the tape and estimates the time needed to reach the block; it then performs services for other users. Just before the estimated time has passed, the routine checks to see if the block has been reached.

If your program attempts to write on a write-locked tape, or to access a DECtape device that has no tape on it, the monitor stops your job and prints (on the terminal) the message:

```
DEVICE DTAu OPERATOR nn ACTION REQUESTED
```

where **u** is the unit number and **nn** is the node number. When the problem is corrected, you can use the CONTINUE command to resume the job (unless an explicit or implicit RESET has been issued for the program).

Your program can initialize a DECtape device on only one channel. Input and output occur on this channel; however, they do not occur simultaneously.

Because a DECtape is a directory device, your program must select a file before it can perform I/O to the DECtape. To select a file on a DECtape, use one of the following calls: LOOKUP, ENTER, USETI, USETO, UGETF, or a function of the FILOP. call.

13.3.1 Monitor Calls for DECtape I/O

The monitor calls of greatest interest in using DECtapes are:

CLOSE	Closes the DECtape file.
ENTER	Creates, writes, supersedes, or (after a LOOKUP) appends to a DECtape file. The special DECtape argument list for the ENTER call is described in Section 13.3.2.
FILOP.	Performs various DECtape functions.
IN,INPUT	Performs normal input from the DECtape, except that the DECtape service routine reads the link in each block to determine the next block to read, and to determine when to set the end-of-file flag (IO.EOF) in the file status word.
LOOKUP	Selects a file for input. The LOOKUP call can be used in deleting, reading, updating, renaming, or appending to a file. A LOOKUP call to a DECtape device uses the special LOOKUP argument list. For more information, refer to Section 13.3.2.
MTREW.	Rewinds the DECtape. This function is also available with the FILOP. call, function .FOMTP.
MTUNL.	Rewinds and unloads the DECtape. This function is also available with the FILOP. call, function .FOMTP.
OUT,OUTPUT	Performs output to the DECtape device. The buffer control block is processed as follows: If the left half of word 2 of the buffer control block contains -1, the DECtape service routine changes it to Ø, thereby terminating the file. If the halfword is positive, the routine uses the value as the block number for the next OUT or OUTPUT call. If the halfword is Ø, the routine assigns the block number for the next OUT or OUTPUT call.
RELEAS	Releases the DECtape channel. The monitor copies its directory for the DECtape (maintained in memory) to the DECtape. Commands that issue implicit RELEAS calls (for example, KJOB) also copy the directory to the DECtape.

RENAME — Renames or deletes the DECtape file. A RENAME call to a DECtape device uses a special argument list for the call. You provide the file name, extension, and creation date. The argument list is described in Section 13.3.2.

UGETF — Returns the next free block of the DECtape. If no ENTER or LOOKUP precedes the UGETF call, the monitor returns -1 in the first word of the argument list. If an ENTER or LOOKUP precedes the UGETF call, the monitor returns the first free block nearest the beginning of the tape (instead of nearest the directory). The monitor also changes the spacing factor from four to two. The spacing factor separates blocks, allowing tape to stop and then restart without having to back up. This function is also available with the FILOP. function .FOGTF.

USETI — Sets the DECtape to the specified block number for next input. To assure that the buffer containing the block number is the next one used for input, either use a single buffer or set the IO.SYN bit in the I/O status word (using SETSTS). This forces the monitor to stop after each IN or INPUT call.

USETO — Sets the DECtape to the specified block number for next output. To assure that the buffer containing the block number is the next one used for output, either use a single buffer or set the IO.SYN bit in the I/O status word (using SETSTS). This forces the monitor to stop after each OUT or OUTPUT call.

NOTE

USETI/USETO calls do not operate with DECtape the same way as they do with disk. On disk, USETO n puts you at block n in the file, regardless of the block's actual position on the disk. On DECtape, USETO n puts you at block n on the tape, regardless of the file that contains that block.

UTPCLR — Clears the DECtape directory. This function is also available with FILOP. function .FOUTP.

On each interrupt, the monitor updates the device status word for the DECtape; use the DEVSTS monitor call to retrieve the status.

13.3.2 Special Argument Lists

The LOOKUP, ENTER, and RENAME monitor calls for a DECtape device use special formats for the argument list for the call.

13.3.2.1 Using LOOKUP with DECtapes - The LOOKUP call selects a file for input. It is used in the process of deleting, reading, changing, renaming, or appending to a file. The calling sequence for a LOOKUP to a DECtape file is:

```
        LOOKUP channo,addr
          error return
        normal return

addr:   SIXBIT/filename/
        SIXBIT/extension/
        Ø
        Ø
```

where: the LOOKUP call accepts a channel number (**channo**) and the address of the argument list (**addr**). The information you must provide and the information returned in the argument block is listed in Table 13-1. The LOOKUP monitor call sets up an input file (specified by **filename** and **extension** in the the argument block) on the specified I/O channel. (Note that you must initialize the channel first, using the OPEN call.)

NOTE

The LOOKUP/ENTER extended argument list can be used with DECtapes. Refer to Chapter 11.

The contents of the argument block are matched against the file names and extensions in the DECtape directory. If the monitor does not find a match, the error return is taken and the monitor returns an error code in the right half of **addr+1**. The error codes are described in Chapter 11. If the file name is matched, the normal return is taken, and the information listed in Table 13-1 is returned in the argument block.

Table 13-1: LOOKUP/ENTER/RENAME Argument Block for DECtape

Word	Bits	Contents	LOOKUP	ENTER	RENAME
Ø	Ø-35	SIXBIT/filename/	A	A	A
1	Ø-17	SIXBIT/extension/	A	A	A
	18-2Ø	High-order 3 bits of the creation date.	V	AØ	AØ
	21-25	Zero			
	26-35	First block number.	V	V	V
2	Ø-23	Zero.			
	24-35	Low-order 12 bits of the creation date.	V	AØ	AØ
3	Ø-17	Negative word length of the zero-compressed file.	V	-	-
	18-35	Core address of the first word of the file minus 1 (obsolete).	-	-	-

A = argument in your program
V = value returned from the monitor
I = ignored.

On a normal return, the first block of the file is found as follows:

- The first 83 words in the DECtape directory block are searched backwards, beginning with the slot immediately before the slot pointing to the directory block. The search procedure stops when the slot containing the desired file number is found.

- The block associated with this slot is read; bits 18-27 of the block's first word (the bits containing the block number of the first block of the file) are checked. If the bits are equal to the block number of this block, then this block is the first block; if not, then the block with that block number is read as the first block of the file.

13.3.2.2 Using ENTER with DECtapes - The ENTER call selects a file for output. It is used in the process of creating, writing, appending to, and superseding files.

The calling sequence for the ENTER call for DECtapes is:

```
ENTER channo,addr
  error return
normal return
```

where addr is the address of the argument block, described in Table 13-1.

The ENTER call requires the channel number (**channo**) and address of the argument list (**addr**). The argument list, the information you must supply, and the information returned by the monitor, are listed in Table 13-1. If you specify an extended argument list, the monitor ignores the extra words and leaves them unchanged.

The monitor searches the DECtape directory for the specified file name and file extension. If the monitor does not find a match, and there is room in the directory, the monitor records information in the DECtape directory. If the monitor finds a match, the specified entry replaces the old entry and the old file is reclaimed immediately. The monitor then records the file information in the DECtape directory.

When the monitor replaces an existing file with a new file, it is superseding a file. Superseding a file on DECtape differs from superseding a file on disk. On DECtape, because of its small size, the space is reclaimed before the file is written instead of after the file is written. That is, once the ENTER has been performed, the old file is deleted.

13.3.2.3 Using RENAME with DECtapes - The RENAME call changes the file name or file extension of a file, or deletes it from the DECtape.

For use with DECtapes, the RENAME calling sequence is:

```
RENAME channo,addr
  error return
normal return
```

In the calling sequence, RENAME requires a channel number (**channo**) and the address of the argument list (**addr**). If you specify an extended argument list (longer than 4 words) the extra words are ignored and left unchanged. Table 13-1 lists the contents of the argument list before the call and after a successful return by the monitor.
Unlike a RENAME on a disk device, a RENAME on DECtape works on the **last** file selected (using LOOKUP or ENTER) on the device, not the last file on the specified channel. The calling sequence required to successfully RENAME a file on DECtape is:

```
LOOKUP channo,addr
RENAME channo,addr1

     or

ENTER channo,addr
RENAME channo,addr1
```

13.4 DECTAPE FORMATS

A DECtape is 260 feet (79.24 m) long and 10 tracks wide. The monitor views the tape as having 577 decimal blocks, numbered from 0 to 1101 (octal). Blocks 1 and 2 are reserved for the bootstrap loader. Block 100 (decimal) is the directory block. The directory contains up to 22 (decimal) files.

Each block of a DECtape can contain 128 36-bit words; the entire tape can contain 73,856 words.

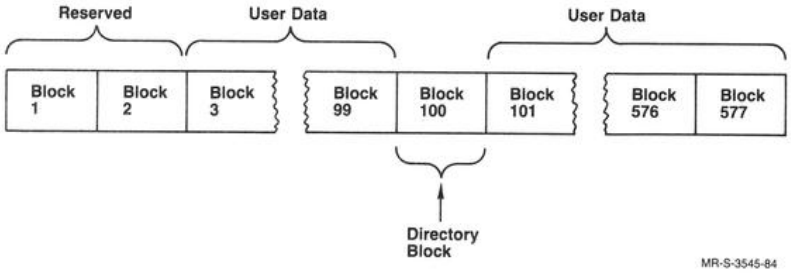


Figure 13-2: DECtape Format

Blocks on a DECtape (see Figure 13-2) are used as follows (block numbers in decimal):

Block	Use
0	Not used. May be read in buffered I/O mode.
1-2	Reserved for bootstrap loader
3-99	Data
100	Directory
101-577	Data

If your program requests a block number larger than 577, the monitor sets the IO.BKT bit in the I/O status word.

13.4.1 Directory Format

The directory block is block number 100 (decimal) and is 128 decimal words long.

A DECtape directory contains the following:

1. A block-to-file index, showing which blocks belong to which files.
2. A list of file names.
3. A list of file extensions.
4. A list of file creation dates.
5. A DECtape label.

Note that, unlike disk files, DECtape files do not have protection codes associated with them.

13.4.1.1 Summary of DECtape Directory Block - Figure 13-3 is a summary of the DECtape directory block. For example, if block number 14 contained file number 10, which was named FILE.MAC, the directory would appear as shown in Figure 13-4.

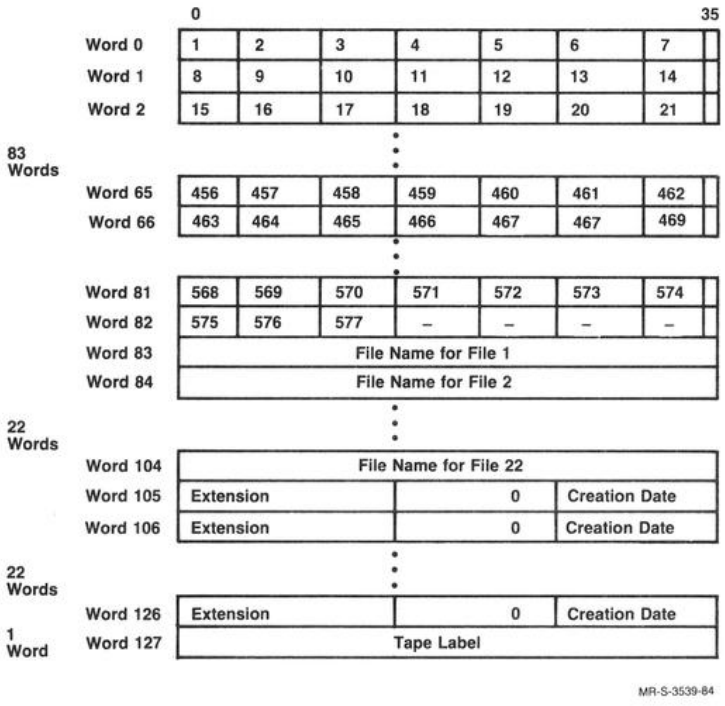


Figure 13-3: DECtape Directory Block

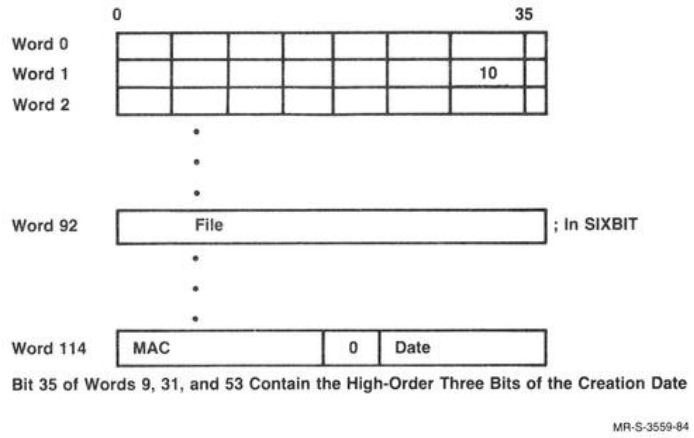


Figure 13-4: Directory Block for FILE.MAC

13.4.1.2 Block-to-File Index - The block-to-file index is in the first 83 words of the directory, words Ø through 82 (see Figure 13-5). Each word contains seven 5-bit file numbers; bit 35 is used otherwise (see creation dates).

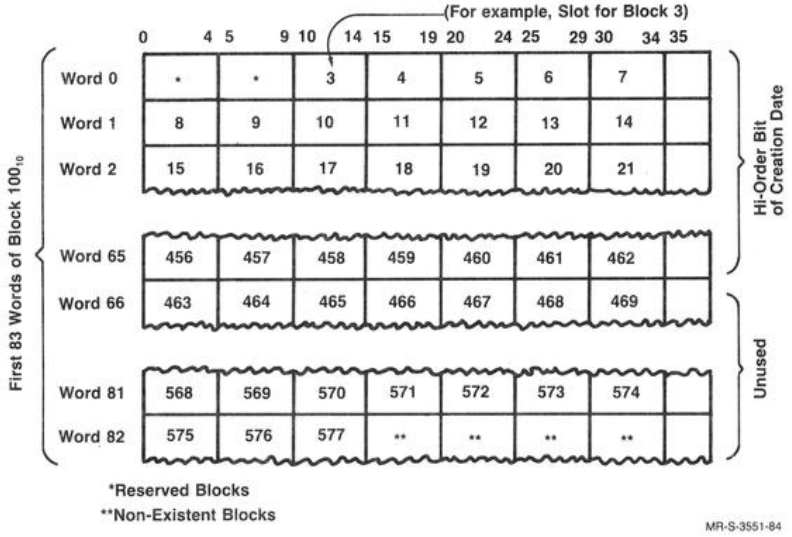


Figure 13-5: First 83 Words on the DECtape of the Directory Block

Each file number in the block-to-file index shows which file owns the corresponding block. For example, the third file number (third byte of directory word Ø) indicates that block 3 of the DECtape contains part of the given file. The 11th file number (fourth byte of directory word 1) indicates that block 11 of the DECtape contains part of the given file.

Note that while a five-bit byte can represent 32 files, there are only 22 files. Code 3Ø is stored in the byte corresponding to the directory block, and code 31 is stored in bytes corresponding to blocks 1, 2, and 578-581. Thus all bytes with Ø on them represent real, existing, valid-to-use, free blocks on the tape. Codes 23-29 are reserved.

Blocks 1, 2, and 100 are not used because they are reserved blocks; blocks 578 through 581 are not used because these blocks do not exist.

13.4.1.3 List of File Names - The list of file names is in words 83 through 104 of the directory (see Figure 13-6). Each word contains the SIXBIT name of a file on the DECtape. This list is the key to the block-to-file index in the directory. A block having the file number 4 in the block-to-file index belongs to the file listed fourth in the list of file names.

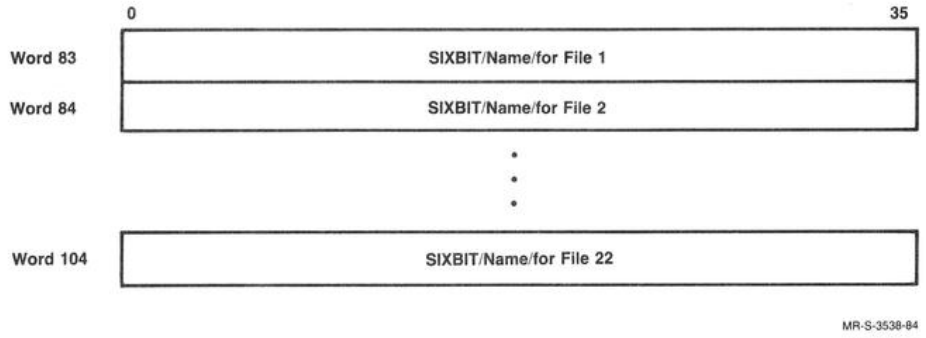


Figure 13-6: Words 83-104 of DECtape Directory

13.4.1.4 List of File Extensions - The list of file extensions is in the left halves of words 105 through 126 of the directory (see Figure 13-7). The first extension is for the first file name in the list of file names, and so forth.

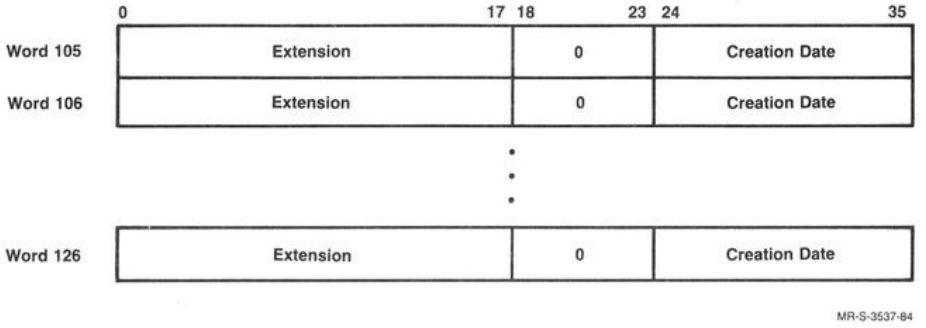


Figure 13-7: Words 105 to 126 of the Directory Block

13.4.1.5 File Creation Dates - The low-order 12 bits of the creation date for each file is in bits 24 through 35 of the same word containing the extension for the file. The high-order 3 bits of the creation dates are stored in the last bits of words 0 through 82 of the directory (see Figure 13-8).

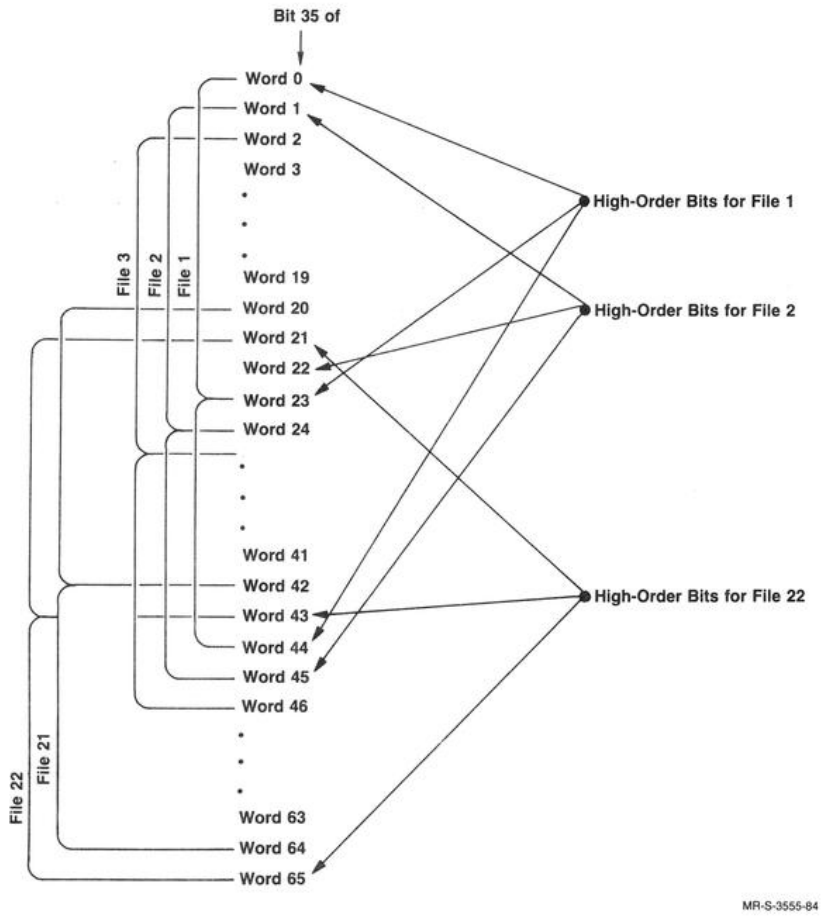


Figure 13-8: High-Order Three Bits of Creation Date

Note that bits 18-23 are apparently free. These bits are not free, however. Old DECtape formats used these as the amount of memory needed to hold the .SAV file (meaningless for non-.SAV files). Thus, because of old tapes, these bits cannot be reused.

For example, the 15-bit creation date for the first file is stored as follows:

Bits	Stored
1	Bit 35 of directory word Ø
2	Bit 35 of directory word 22
3	Bit 35 of directory word 44
4-15	Bits 24-35 of directory word 1Ø5

More generally, the 15-bit creation date for file number **n** is stored as follows:

Bits	Stored
1	Bit 35 of directory word n-1
2	Bit 35 of directory word n+21
3	Bit 35 of directory word n+43
4-15	Bits 24-35 of directory word n+1Ø4

13.4.1.6 DECtape Label - A SIXBIT label for the DECtape is stored in word 127 of the directory.

13.4.2 Data Block Format

A data block on a DECtape (see Figure 13-9) has a link word and up to 127 (decimal) data words. All user data blocks on a DECtape have this format. The link word is in the format:

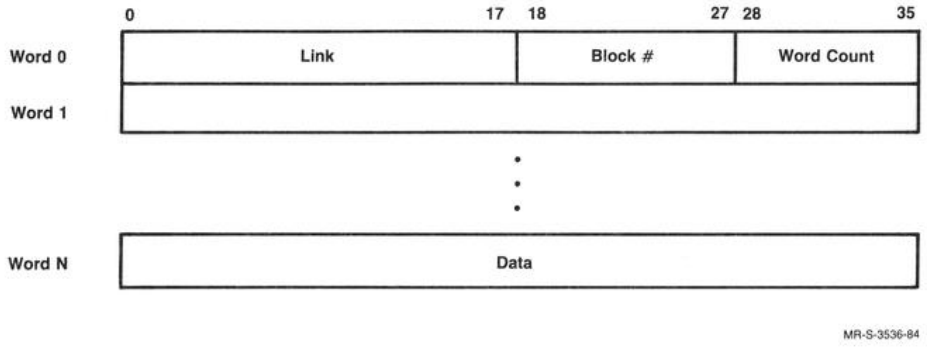


Figure 13-9: Data Block Format

Bits	Contents
Ø-17	Block number of next block in file
18-27	Block number of first block in file
28-35	Number of data words in block

If Bits Ø through 17 contain Ø, the current block is the last in the file.

The DECtape service routine allocates the first free block that is before and closest to the directory block; that is, the routine begins reading backwards at block 99 and allocates the first free block it finds. When the routine encounters the end of the tape, it reverses direction.

The DECtape service routine does not write blocks contiguously; each allocated block is separated by a spacing factor, which allows the tape to stop and start again without having to back up. The spacing factor is normally four blocks; if your program uses dump mode and issues a UGETF monitor call followed by an ENTER call, the spacing factor is set to two.

13.5 DECTAPE I/O STATUS

The I/O status bits for DECtape are as follows:

Bits	Symbol	Meaning
18-21	IO.ERR	Error flags:

Flag	Symbol	Error
18	IO.IMP	Tape being read/written in improper mode.
19	IO.DER	Data was missed on the tape.
20	IO.DTE	Parity error.
21	IO.BKT	Block too large or too small, or the tape was full when the OUT or OUTPUT call was issued.
22	IO.EOF	End-of-file reached.
23	IO.ACT	Device active.
28	IO.SSD	Semistandard mode.
29	IO.NSD	Nonstandard mode.
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
0	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
10	.IOIMG	Image mode.
13	.IOIBN	Image binary mode.
14	.IOBIN	Binary mode.
16	.IODPR	Dump record mode.
17	.IODMP	Dump mode.

You can set the I/O status bits IO.SSD and IO.NSD and the data mode in your OPEN or INIT call for the DECtape I/O channel. You can also set these I/O status bits with SETSTS. Bits 18 through 23 of the I/O status word, however, are set by the monitor on an error return from a data transmission call (IN, INPUT, OUT, or OUTPUT) to indicate the reason for the error return.

CHAPTER 14

MAGTAPES (MTA)

An unlabelled magtape is a sequential I/O, nondirectory device. The data on the tape begins with a beginning-of-tape (BOT) mark. Each file on the tape ends with an end-of-file (EOF) mark. The last file ends with an end-of-tape (EOT) mark, which is two EOF marks. The physical end of the tape also has a mark: a metal reflector attached to the tape.

A file on a magtape contains one or more records; each record occupies one or more blocks. If the last block used by a record is not filled by the record, a gap is left unused (up to the end of the block).

Magtapes cannot be connected to MPX (multi-plexed) channels.

TOPS-10 supports both 7-track and 9-track magtape devices. A 9-track tape device can be operated in either DIGITAL-compatible or industry-compatible mode.

A 9-track tape can be either labelled or unlabelled. A tape label identifies, defines, and protects the data on the tape. A labelled magtape can be accessed using LOOKUP and ENTER calls, like a directory device. For a discussion of the format of tape labels, see the TOPS-10 Tape Processing Manual.

The default density of tapes is set by the system administrator when the monitor is generated by MONGEN. You can set a different density for a tape by using the SET DENSITY command, or by setting the bit IO.DEN in the file status word by using either the SETSTS monitor call, or by using the TAPOP. monitor call with the .TFDEN function.

Magnetic tape density can be one of the following:

200 bits/inch (8.1 rows/mm)
556 bits/inch (22.5 rows/mm)
800 bits/inch (32.2 rows/mm)
1600 bits/inch (65.3 rows/mm)
6250 bits/inch (225.5 rows/mm)

The default buffer size for a magtape device is 128 decimal words.

A magtape transfer is limited to 7681 words for KS systems, and, for KL systems, is limited according to the type of controller as listed here and in the TAPUUO source module.

Controller	Limit (in words)
TXØ1	128K
DX2Ø	16K
TMØ2	16K
TMØ2/RH11	16K
TM78	16K

Specific information about magnetic tape usage, as appropriate to each type of magtape, is given in the TOPS-1Ø Operator's Guide.

14.1 MAGTAPE DEVICE NAMES

A magtape device has a name of the form:

MTcnnu

where c is the controller name: A, B, C, and so forth; nn is the node number of the CPU; and u is a unit number in the range Ø to 7.

The magtape controllers and the magtape units that are supported for TOPS-1Ø are listed in the TOPS-1Ø Operator's Guide.

14.2 MAGTAPE DATA MODES

A magtape device can use any of the following data modes: ASCII, ASCII line, byte, image, image binary, binary, dump record, or dump mode. The default buffer size for buffered modes is 2ØØ (octal) words; you can change the buffer size by using the SET BLOCKSIZE command, or by using the TAPOP. .TFBSZ function.

1. ASCII mode reads or writes ASCII characters exactly the same in the buffer and on the tape. Parity checking is done by the magtape system, not by the monitor.

2. ASCII line mode is identical to ASCII mode.

3. Byte mode transfers bytes between buffer and tape according to a byte pointer and byte count. For output, the monitor computes the byte count and reads the byte size from the buffer ring control block. Each byte is written as a tape record (also called the "tape frame"). For input, the monitor obtains the byte count from the magtape service routine and reads the byte size from the buffer ring control block. The default byte size is 8 bits, where the low-order 4 bits of each 36-bit word are lost.

NOTE

In other modes, whole words of 36 bits are transferred regardless of the actual number of bytes in the buffer (exception: see TAPOP. function .TFMFC). In byte mode, only the actual number of data bytes are transferred.

4. Image mode is identical to ASCII mode, except that data is taken as 36-bit bytes.

5. Image binary mode is identical to image mode.

6. Binary mode is identical to image mode.

7. Dump record mode is unbuffered and uses a command list to transfer data between memory and the magtape device. The monitor call that requests the magtape I/O gives the address of the command list, such as an IN, INPUT, OUT, or OUTPUT call.

   The default block size is 200 (octal) words; you can change the record size by using the SET BLOCKSIZE command or the TAPOP. .TFBSZ function.

   A command list consists of words of one of the following forms:

```
IOWD buflength,buffer
XWD 0,nextcmd
Z
```

   where the IOWD format calls for the number of words given by **buflength** to be transferred between the magtape device and memory beginning at **buffer**; the XWD format directs that the command list continues at **nextcmd**; and the Z (zero) word ends the command list.

   On input, each IOWD command begins a new buffer in memory. If the record read is too short for the buffer, input continues in the next record on the tape; if the record read is too long, the IO.BKT bit in the file status word is set and the monitor takes the error return from the relevant monitor call.

   On output, each IOWD command begins a new record on the magtape. If the buffer is too long for the record length, the data continues to new records on the tape, as required; if the buffer does not fill the last record, it remains a short record on the tape.

   On an I/O error or physical end-of-tape, the magtape service routine ends I/O and stops reading from the command list. On an end-of-file input, the input call takes the error return and the IO.EOF bit is set in the file status word. On the next input call, the next record (in the next file) is read from the tape.

   The read backwards function is not allowed in dump record mode.

8. Dump mode is unbuffered and transfers variable-length records between the magtape device and memory. Dump mode uses a command list identical to that of dump record mode.

Unlike DUMP RECORD mode, DUMP mode processes exactly one tape record for each IOWD. If an INPUT operation encounters a record that is shorter than that specified in the IOWD, the remainder of the memory buffer is not changed. On output, the records are not split up, and the length of each record written to the tape is exactly that specified in the IOWD.

14.3 MAGTAPE I/O

A number of monitor calls can be used for magtape I/O. These are:

DEVSTS	Reads file status bits.
ENTER	Opens a labelled magtape for writing.
FILOP.	Performs directory device functions for labelled magtape.
LOOKUP	Opens a labelled magtape for reading.
MTAID.	Defines a tape reel identifier for a magtape device.
MTBLK.	Writes three inches of blank tape.
MTBSF.	Skips backward over one file on tape.
MTBSR.	Skips backward one record on tape.
MTCHR.	Returns characteristics for a magtape device.
MTDEC.	Initializes for DIGITAL-compatible 9-track tape.
MTEOF.	Writes an end-of-file mark on tape.
MTEOT.	Skips forward to end-of-tape.
MTIND.	Initializes for industry-compatible 9-track tape.
MTLTH.	Sets tape to read next record at low threshold (TM10 only).
MTREW.	Rewinds the tape.
MTSKF.	Skips forward over one file on tape.
MTSKR.	Skips forward one record on tape.
MTWAT.	Waits for tape I/O to complete.
TAPOP.	Performs any of the above functions and many other functions for a tape.

14.4 MAGTAPE I/O STATUS

The I/O status bits for magtapes are listed below.

Bits	Symbol	Meaning
18-21	IO.ERR	Error flags. If all these bits are set, the error was detected by the tape label handler; use the DEVOP. call to determine the extended error code.
22	IO.EOF	End-of-file mark found. You must clear this bit using SETSTS before reading further, or CLOSE to reset the tape status.
23	IO.ACT	Device active.
24	IO.BOT	Beginning of tape.
25	IO.EOT	Physical end of tape was encountered. Physical EOT is several feet before the actual end of tape. It is seen when passed over on a write operation as an error return with this bit set in the file status word. You must clear this bit and condition (using the SETSTS call) before another write operation can be successful. Physical EOT is not seen on read operations. Physical EOT does not prevent your program from writing off the end of tape. If your program does not heed the physical EOT warning, it is allowed to keep writing. Logical EOT does not prevent your program from reading past it. Your program must check for two consecutive EOF marks, which indicates the logical EOT.
26	IO.PAR	Parity: Ø = even (BCD only) 1 = odd
27-28	IO.DEN	Density: Ø = standard 1 = 2ØØ BPI 2 = 556 BPI 3 = 8ØØ BPI

Flag	Symbol	Error
18	IO.IMP	Output attempted on a write-locked unit, or an illegal operation was attempted.
19	IO.DER	Data was missed on the tape or the tape was bad, or the transport is hung.
2Ø	IO.DTE	Parity error.
21	IO.BKT	The record read from the tape is too long for the buffer.

Other densities (16ØØ, 625Ø) must be specified by setting IO.DEN to Ø on an OPEN, and then using a TAPOP. to set the desired density.
Odd parity is preferred. The user program should specify even parity only when creating a tape to be read in binary coded decimal (BCD) on another type of computer system.

Bits	Symbol	Meaning
29	IO.NRC	Read without reread check.
32-35	IO.MOD	Data mode. Defines method for writing data into buffers. The modes are:

Code	Symbol	Meaning
Ø	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
3	.IOBYT	Byte mode.
1Ø	.IOIMG	Image mode.
13	.IOIBN	Image binary mode.
14	.IOBIN	Binary mode.
16	.IODPR	Dump record mode.
17	.IODMP	Dump mode.

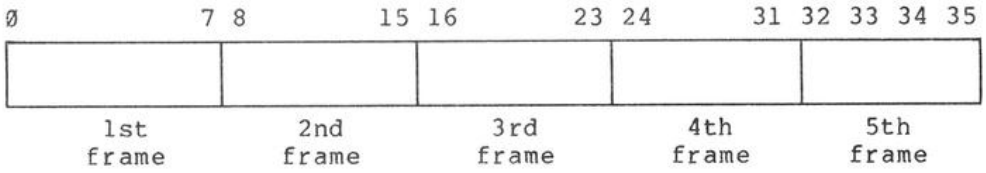
14.5 MODES SET BY .TFMOD

The magnetic tape data formats, which can be set with TAPOP. function .TFMOD (Code 1ØØ7/2ØØ7) are described in this section. These hardware data modes describes how data is stored on a magnetic tape.

The following sections describe which bits are stored in which tracks and how many frames are required to store a word of data. Refer to the description of the TAPOP. monitor call in Chapter 22 for a description of how to set the desired magnetic tape data mode. The diagrams in this section are logical representations of data on a magnetic tape. In actuality, the parity frame is in the center of the tape, and the order of the other frames is not necessarily as pictured.

Code Ø (.TFMDD) sets DEC-compatible core dump mode for either 7-track or 9-track magnetic tapes (MTAPE 1ØØ also sets this mode.) The type chosen depends on the magnetic tape unit used. This data mode function code is the equivalent of code 1 (.TFMID) for 9-track tapes and code 5 (.TFM7T) for 7-track tapes. On a 9-track tape, DEC-compatible core dump mode stores one 36-bit word of data in five frames. Note that the last frame is only half-used. One word actually requires 4 and one-half frames.

Each data word in 9-track DEC-compatible core dump mode is formatted as shown below.



For each data word in memory, there are five magnetic tape bytes per 36-bit word, with parity bits unavailable to the user. Bits are read and written on the tape as shown below.

Table 14-1: 9-Track DEC Dump Mode

Tracks

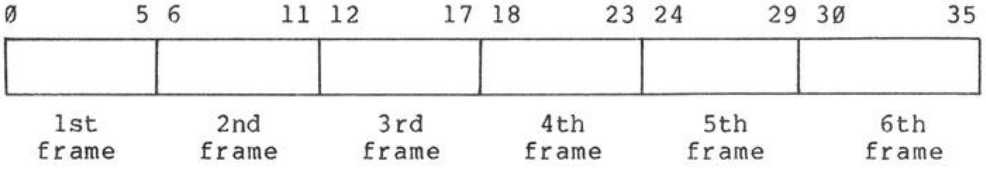
9	8	7	6	5	4	3	2	1	
Ø	1	2	3	4	5	6	7	P	
8	9	1Ø	11	12	13	14	15	P	
16	17	18	19	2Ø	21	22	23	P	Frames
24	25	26	27	28	29	3Ø	31	P	
Ø	Ø	Ø	Ø	32	33	34	35	P	

When writing on TM1Øs, bits 3Ø and 31 are written twice. First they are written as shown in the diagram above and they are then copied in tracks 7 and 6 of byte 5. Tracks 9 and 8 of byte 5 remain zero. When writing with a DX1Ø, TMØ2-C, or TM1Ø-C, bits 3Ø and 31 are written only once. Tracks 9, 8, 7, and 6 contain zero.

When reading from a TM1Ø, parity bits and tracks 9 and 8 of frame 5 are ignored. Frame 4, tracks 2 and 3 are ORed with Frame 5, tracks 6 and 7. These are bits 3Ø and 31 of the data word. On a DX1Ø, TMØ2-C, and TC1Ø-C, the parity bits, frames 9, 8, 7, and 6 are ignored.

Code 1 (.TFMID) sets 7-track core dump mode, one 6-bit byte is stored in each frame of the tape. Six frames are required to store one 36-bit word. This mode is the only hardware mode supported for 7-track magnetic tapes.

Each data word in 7-track DEC-compatible core dump mode is formatted as shown below.



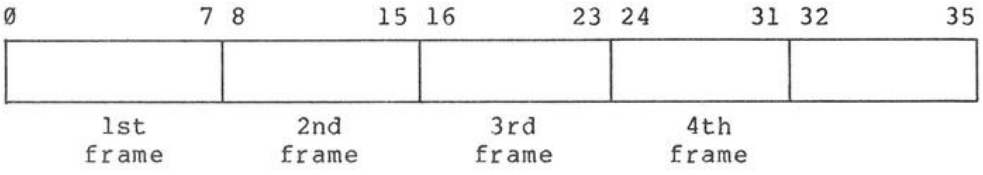
Bits are read and written on the tape as shown below.

Table 14-2: 7-Track Dump Mode

Tracks							
7	6	5	4	3	2	1	
Ø	1	2	3	4	5	P	
6	7	8	9	1Ø	11	P	
12	13	14	15	16	17	P	Frames
18	19	2Ø	21	22	23	P	
24	25	26	27	28	29	P	
3Ø	31	32	33	34	35	P	

Code 2 (.TFM8B) sets industry-compatible core dump mode for 9-track tapes. This mode is compatible on any machine that reads and writes 8-bit bytes (for example, System 36Ø/37Ø and PDP-11s). When a read operation is performed, four frames (8-bit bytes) are read into each word (left-justified). The TMØ2 and TM1Ø-C in industry compatible mode at 8ØØ bpi in bits 32-35 give an indication of a parity error in the corresponding frame of a word. In industry compatible mode at 16ØØ bpi, bits 32-35 are indeterminate. The DX1Ø and TC1Ø-C, return zeroes in these four bits. The information read into bits 32 through 35 is not user data. On a write operation, the left most four 8-bit bytes of each word are written out in four frames; the remaining 4 rightmost bits (that is, bits 32 through 35) are ignored. Only bits Ø through 31 are written onto the tape.

Each data word in 9-track Industry-compatible core dump mode is formatted as shown below.



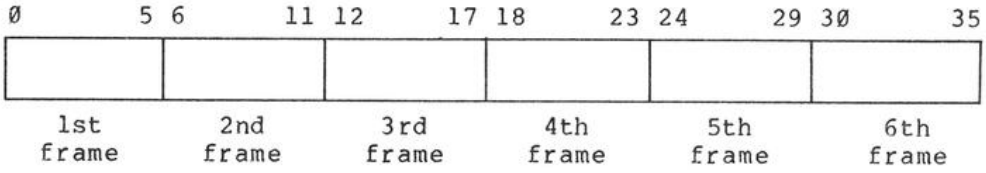
Bits are read and written on the tape as shown below.

Table 14-3: 9-Track Industry-Compatible Dump Mode

	Tracks								
	9	8	7	6	5	4	3	2	1
	Ø	1	2	3	4	5	6	7	P
	8	9	1Ø	11	12	13	14	15	P
Frames	16	17	18	19	2Ø	21	22	23	P
	24	25	26	27	28	29	3Ø	31	P

When using industry-compatible mode and buffered I/O, your program should insert the byte size in the byte pointer before issuing the first IN, INPUT, OUT, or OUTPUT monitor call.

Code 3 (.TFM6B) sets SIXBIT mode for 9-track magnetic tapes. This mode is only available on TU7x - series drives. The format of the data word is shown below.



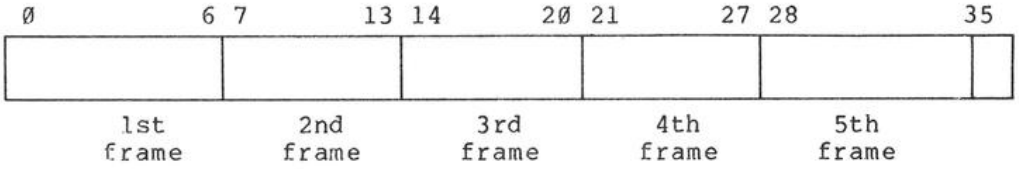
Bits are read and written on the tape as shown below.

Table 14-4: 9-Track SIXBIT Mode

	Tracks								
	9	8	7	6	5	4	3	2	1
	Ø	Ø	Ø	1	2	3	4	5	P
	Ø	Ø	6	7	8	9	1Ø	11	P
	Ø	Ø	12	13	14	15	16	17	P
Frames	Ø	Ø	18	19	2Ø	21	22	23	P
	Ø	Ø	24	25	26	27	28	29	P
	Ø	Ø	3Ø	31	32	33	34	35	P

Code 4 (.TFM7B) sets 7-bit mode, called ANSI ASCII (not available on TM10s and TC10-Cs). This mode stores one 7-bit ASCII byte in each frame of the tape. This mode is useful for transferring ASCII data from DECsystem-10s to 8-bit byte-oriented machines, such as PDP-11s and System 360/370s. Five left-justified (in core) 7-bit bytes are stored in five frames on the magnetic tape. Bit 35 must be zero to conform to ANSI standards. Bit 35 is written into the high-order bit of the last frame of each word. The other high-order bits are set to zero on write operations. When the tape is read, all five high-order bits are ORed and the result is stored in bit 35.

Each data word in ANSI ASCII mode is formatted as shown below.



Data is read and written on the tape in the following format.

Table 14-5: ANSI ASCII Mode

	Tracks								
	9	8	7	6	5	4	3	2	1
	0	0	1	2	3	4	5	6	P
	0	7	8	9	10	11	12	13	P
	0	14	15	16	17	18	19	20	P
Frames	0	21	22	23	24	25	26	27	P
	35	28	29	30	31	32	33	34	P

14.6 READ BACKWARDS (TX01, TM02, AND TX02 ONLY)

When reading a tape backwards, the monitor reads data forwards (i.e., inverted) into the buffer. If the buffer is larger than the record, a zero-fill will result at the beginning of the buffer. For example:

BUFSIZ - WRDCNT = first word of data

If your program contains a multiple IOWD list your program must reverse the I/O bit to enable read forward. Reading backwards into BOT will set the EOF and BOT bits. Note that a tape cannot be read backwards if mode 16 or labelled tapes are used.

Read backwards must be set after the INIT, OPEN, or FILOP. monitor call, because the read backwards function is cleared on an OPEN, INIT, FILOP., or RELEAS monitor call.

When reading backwards, the records are returned to the user in reverse order, but the bytes within each physical record are returned in forward order. Therefore, to prevent the appearance of scrambled data when reading backwards, you should read the records on physical boundaries only.

14.7 PROGRAMMING I/O TO LABELLED MAGTAPES

By default, a magtape contains only the information described in Section 14.5. However, GALAXY 4.1 batch and spooling system provides the opportunity to write labelled magnetic tapes, which can be accessed in a manner similar to a directory device such as disk and DECtape.

With unlabelled magtapes, the LOOKUP and ENTER monitor calls are not operational. They are ignored to provide your program with device independence. (Note that your program will always skip when these calls are ignored.) However, the calls are used with directory devices and labelled magtapes.

Label parameters are temporarily stored in the monitor's data base for the tape drive until the first input or output operation is performed on the tape drive. When I/O is done, the label parameters are transferred to the tape or made available to the program, depending on the direction of I/O. An ENTER call loads the parameters from the argument block onto the tape. A LOOKUP call loads the parameters from the tape into the argument block. A LOOKUP with no file name or extension will return information about the current file on the tape. Both the four-word and extended argument blocks for LOOKUP/ENTER are allowed.

For a labelled magtape, ENTER loads the label parameter block, and a LOOKUP reads the label parameter block, to find the requested file by file name and extension. This functionality is also provided by the TAPOP. monitor call with function .TFLPR.

Magnetic tape label processing is not fully supported by the TOPS-10 monitor. Therefore, if you issue a monitor call that reports your position on the tape, the number of files will include the number of label record files (2 per data file). As a result, the MTCHR. UUO and the .TFSTA function of TAPOP. UUO reports the number of data files times 3.

The following information is loaded from the label parameter block of a labelled magtape. For more information about the contents of each word, refer to the description of the .TFLPR function of TAPOP. in Chapter 22.

Word	Byte	Contents
.TPREC:	TR.FCT	Forms control
	TR.RFM	Record format
.TPRSZ		Record size
.TPBSZ		Block size
.TPEXP:	TP.ECR	Creation date
	TP.EEX	Expiration date
.TPPRO		Protection code
.TPSEQ		File sequence number
.TPFNM		File name and extension
.TPGEN:	TP.GEN	Generation
	TP.VER	Version

The following defaults for the label parameters will be set by an ENTER to the tape label:

Parameter	Default
Forms Control	No forms control. Same as .TFCNO code in argument block for function .TFLPR of TAPOP. call.
Record Format	Fixed record format. Same as .TRFDF in argument block for function .TFLPR of TAPOP. call.
Record Size	Buffer size times number of bytes per word. Buffer size is reported by DEVSIZ monitor call. Number of bytes per word depends on the format used to write the tape (refer to Section 14.5).
Block Size	Default is the buffer size.
Creation date	Defaults to "today" (current date).
Expiration date	Defaults to "today."
Protection code	No default. Taken from ENTER argument block.
File Sequence No.	Every file on the tape is assigned a sequential sequence number as it is written to the tape.
File name	Both file name and extension are taken from ENTER argument block. If file name in argument block is Ø, FILE.nnn is used, where nnn is the file sequence number.
Generation No.	Defaults to Ø.
Version No.	Defaults to Ø.

CHAPTER 15

TERMINALS (TTY) AND PSEUDO-TERMINALS (PTY)

15.1 TERMINAL DEVICE NAMES

The name of a terminal is of the form **TTYnnn**, where **nnn** is a 3-digit octal number (leading zeroes may be omitted).

Local terminals (connected to the RSX-20F front end on the KL or directly to the KS) will always have the terminal name that contains the number of the terminal line. Therefore, the terminal connected to line 27 will always have the name TTY027. However, terminals connected through an ANF-10 remote station will have a terminal name based on the node name, such as NOVA_22, where the node name is NOVA; or 32_22, where the ANF-10 node number is 32 and the line number is 22. When a remotely-connected terminal is connected to the system, the monitor assigns it a terminal name on a dynamic basis, from a pool of unassigned terminal names. Therefore, when NOVA_22 is connected to the system, the monitor assigns it a terminal name, such as TTY150. If the terminal is disconnected and connected again, it may be assigned a different terminal name, such as TTY56.

15.2 TERMINAL DATA MODES

A terminal uses a buffer size of 23 (octal) words. A terminal can use any of the following buffered data modes:

- ASCII mode transmits 7-bit characters packed left justified, five characters per word. If you attempt to do input from the terminal in ASCII mode, the monitor will use ASCII line mode.

- ASCII line mode receives 7-bit characters packed left-justified, 5 characters per word. When doing input from the terminal, the monitor will return a buffer to your program when either the buffer is filled (132 characters) or if a break character (such as linefeed) is input. If you attempt to do output in ASCII line mode, the monitor will use ASCII mode instead. To do the equivalent of ASCII line mode on output, your program must issue OUT or OUTPUT calls itself, after putting each break character in the buffer. See Section 15.4 for more information about break characters. ASCII mode and ASCII line mode are essentially equivalent, in that each line (up to a break) uses a buffer, even though the buffer might be capable of storing more characters. Note that, with IO.BKA set, each character initiates a break condition.

- 8-bit ASCII mode receives 8-bit characters, packed left-justified, in four 8-bit bytes. 8-bit ASCII mode is legal for pseudo-terminals (PTY).

- Image mode is allowed only for terminal I/O and is not allowed for pseudo-terminals. Image mode makes any sequence of input characters legal. Because image mode is a "literal" mode, the terminal settings to prompt processing by the monitor (such as FORMFEED, TABS, and so forth) are not effective in image mode. Carriage-return does not generate an automatic line-feed, line editing is not performed, and control characters, like CTRL/C and CTRL/Z, do not halt the program.

  To avoid having a terminal get hung in image mode (when no input can change the input mode), the monitor simulates an end-of-file if no characters are input for ten seconds. After another ten seconds, the monitor simulates a CTRL/C to release the terminal from image mode.

  The simulation of these CTRL/Zs and CTRL/Cs can be prevented by sending any output to the terminal before the ten seconds have passed. (If no output is needed, you can output a null.)

- Packed image mode (PIM) is allowed for terminal I/O and for pseudo-terminals (PTYs) in full-SCNSER mode. Packed image mode allows efficient throughput of data between programs and terminals; this efficiency is accomplished by minimizing the monitor's manipulation and testing of characters. PIM does not echo terminal input, and the OPEN bit IO.SUP is ignored in this mode.

  A packed-image character is stored in the buffer as an 8-bit value (7-bit ASCII plus parity bit); four characters are stored in each word. Your program can define a "break set" consisting of from one to four break characters for each line initialized in packed image mode. These break characters are defined by using the TRMOP. monitor call. Note that your break set may take into account the parity setting.

  When the monitor receives a break character from the terminal, the character is placed in the buffer and the controlling program is awakened. Other characters are placed into the buffer without waking the program.

  If you define a null break set (Ø), your controlling program is awakened whenever input is available. All available input is returned, as in image mode. Note, however, that CTRL/S and CTRL/Q will stop/start output to the terminal in this mode; this avoids getting the terminal hung in packed image mode.

15.3 TERMINAL CHARACTER HANDLING

The ASCII character set consists of 128 7-bit character codes. The 8-bit ASCII character set has 256 8-bit character codes. Table 15-1 shows the 7-bit ASCII codes and their display characters, and describes any special handling for terminal I/O. The monitor interprets some control characters as line-editing commands. The TOPS-1Ø Operating System Commands Manual describes monitor command line editing.

Table 15-1: Terminal Handling of ASCII Characters

Code	Prints	Name and Special Terminal Handling
ØØØ		Null character ignored on input, suppressed on output (except for image modes).
ØØ1	^A	CTRL/A acts as a CTRL/C for OPSER subjobs and MIC.
ØØ2	^B	CTRL/B acts as a break character for MIC.
ØØ3	^C	CTRL/C returns the job to monitor command level, if issued while the program is waiting for input. All current type-ahead is cleared. When used while the program is doing output to the terminal, CTRL/C acts like CTRL/U, clearing the current line. Two successive CTRL/Cs will return the job to monitor command level.
ØØ4	^D	CTRL/D sends an EOT character (ASCII ØØ4) to the monitor. If this same value were returned to your terminal it might cause certain types of modems to hang up and your terminal connection would be lost. To prevent this from happening, CTRL/D is echoed as ^D (136,1Ø4). Note that CTRL/D acts as an unsolicited DDT/EDDT breakpoint when the SET DDT/EDDT BREAKPOINT facility is enabled (refer to the TOPS-1Ø Operating System Commands Manual).
ØØ5	^E	CTRL/E.
ØØ6	^F	CTRL/F.
ØØ7		CTRL/G rings the terminal bell instead of echoing, and is a default break character.
Ø1Ø		CTRL/H echoes as a backspace and functions as a DELETE (ASCII 177). The backspace is not passed to your program unless it is in APL or a special editor mode.
Ø11		CTRL/I echoes as a tab or an equivalent number of spaces. See SET TTY TAB in the Commands Manual.
Ø12		CTRL/J echoes as a linefeed and is a default break character.
Ø13		CTRL/K echoes as a vertical tab (default break character) or four linefeeds. See SET TTY FORM in the Commands Manual.
Ø14		CTRL/L echoes as a formfeed (default break character) or eight linefeeds. See SET TTY FORM in the Commands Manual.
Ø15		CTRL/M echoes as a carriage-return/line-feed and is passed to the program as a carriage return and linefeed; the linefeed is a default break character. Note: if the terminal is in papertape input mode (SET TTY TAPE), the carriage return is merely passed to the program.
Ø16	^N	CTRL/N.
Ø17	^O	CTRL/O echoes, but is not passed to the program. CTRL/O inverts the terminal output suppression bit, allowing output to be turned on and off. The suppression bit is cleared by a CTRL/C input character and by all of the following monitor calls: IN, INPUT, OPEN, INIT, FILOP., DDTIN, INCHRW, INCHRS, INCHWL, INCHSL, SKPINC, SKPINL and the TRMOP. equivalents.

Table 15-1: Terminal Handling of ASCII Characters (Cont.)

Code	Prints	Name and Special Terminal Handling
Ø2Ø	^P	CTRL/P acts as a proceed character for MIC.
Ø21	^Q	CTRL/Q echoes unless SET TTY XONXOFF is set, in which case it continues output stopped by CTRL/S. Note: CTRL/Q starts papertape if SET TTY TAPE is set. See the SET TTY XONXOFF and SET TTY TAPE commands in the Commands Manual.
Ø22		CTRL/R does not echo, but causes the current terminal line to be retyped. This includes any edits to the line. Note: in RTCOMP mode, CTRL/R is a break character, and does not type the line. See the SET TTY RTCOMP command in the Commands Manual.
Ø23	^S	CTRL/S echoes unless SET TTY XONXOFF is set, in which case it stops terminal output. The output is not lost, as it is for CTRL/O, but waits for CTRL/Q. Note: if the terminal is in papertape mode, CTRL/S stops the tape.
Ø24		CTRL/T does not echo unless SET TTY RTCOMP is set (see the Commands Manual). If SET TTY RTCOMP is set, it echoes as ^T and is a break character. CTRL/T (not in RTCOMP mode) is equivalent to the USESTAT command, which displays job status and timing information.
Ø25	^U	CTRL/U deletes the current terminal input line back to the last break character. CTRL/U echoes as CTRL/U followed by a carriage return and linefeed. On video terminals, ^U erases the current line on the screen. Note: for special editors, CTRL/U is passed to the editor.
Ø26	^V	CTRL/V.
Ø27	^W	CTRL/W deletes one word from the terminal input line. On video terminals, the word is erased from the screen. For special editors, CTRL/W is passed to the editor.
Ø3Ø	^X	CTRL/X.
Ø31	^Y	CTRL/Y.
Ø32	^Z	CTRL/Z acts as an end-of-file character for terminal input, and is a default break character. CTRL/Z echoes as CTRL/Z followed by a carriage return and linefeed but only the CTRL/Z is passed. Many system CUSPs recognize CTRL/Z as an EXIT command.
Ø33	$	CTRL/[echoes as a dollar sign ($) and is the ESCAPE character. ESCAPE is a default break character (see also codes 175 and 176.)
Ø34	^\	CTRL/\ is control-backslash, and when typed on the CTY of KL and KS systems does not echo, but causes the terminal to be connected to the front-end command processor.
Ø35	^]	CTRL/].
Ø36	^^	CTRL/^.
Ø37	^_	CTRL/underscore.
Ø4Ø	(space)	Blank character.
Ø41	!	Exclamation point.
Ø42	"	Quotation mark.
Ø43	#	Number sign.
Ø44	$	Dollar sign.
Ø45	%	Percent sign.

Table 15-1: Terminal Handling of ASCII Characters (Cont.)

Code	Prints	Name and Special Terminal Handling
046	&	Ampersand.
047	'	Apostrophe.
050	(	Left parenthesis.
051	)	Right parenthesis.
052	*	Asterisk.
053	+	Plus sign.
054	,	Comma.
055	-	Minus sign (hyphen).
056	.	Period (decimal point).
057	/	Slash.
060	0	Zero.
...	...	Intervening numerals.
071	9	Nine.
072	:	Colon.
073	;	Semicolon.
074	<	Left angle bracket.
075	=	Equal sign.
076	>	Right angle bracket.
077	?	Question mark.
100	@	At sign.
101	A	Uppercase A.
...	...	Intervening uppercase letters.
132	Z	Uppercase Z.
133	[	Left square bracket.
134	\	Backslash.
135	]	Right square bracket.
136	^	Up-arrow.
137	_	Underscore.
140	`(grave)	Grave accent.
141	a	Lowercase a.
		NOTE Lowercase letters are translated to uppercase unless SET TTY LC is set or the appropriate terminal type has been specified. Refer to the Commands Manual.
...	...	Intervening lowercase letters.
172	z	Lowercase z.
173	{	Left brace.
174	|	Vertical line.
175	}	Right brace. This is converted to ASCII 033 (ESCAPE), if altmode conversion is enabled.
176	~	Tilde. This is converted to ASCII 033 (ESCAPE) if altmode conversion is enabled (using SET TTY ALTMODE command).
177		DELETE deletes a character from the current terminal input line. DELETE is ignored in papertape mode, and is passed to DDT and special editors.

15.4 BREAK CHARACTER SET

A set of control characters acts as a default set of break characters. A break character defines the end of line for an INCHWL call.

The default break character set is:

CTRL/G
CTRL/J
CTRL/K
CTRL/L
CTRL/Z
ESCAPE

In addition, if the terminal has SET TTY RTCOMPATIBILITY set, CTRL/R and CTRL/T act as break characters.

If the terminal is in SLAVE mode, the following characters are also break characters:

CTRL/C
DELETE
CTRL/U
CTRL/W
CTRL/R
CTRL/T

You can define your own set of break characters by setting the I/O status bit IO.ABS in the OPEN call, and including a bit mask in the argument to the OPEN call for this channel. The IO.ABS state can be set or cleared at any time using the SETSTS call. When set, the state is controlled using the TRMOP. call. When IO.ABS is initially set, the break mask will be set to all zeros, implying that no characters are defined as break characters, and the field width is initially set to one, causing normal echo of control characters and a wakeup with each character that is typed.

You handle input with INCHWL, the appropriate TRMOP. function (input line), or with buffered terminal input. Each field (line) will be determined by the break set or the field width. Wakeup can occur prematurely, such as when monitor buffer space is limited. You should ensure that your program can handle early breaks.

The call can be used to read the break set as well as write it. On a read function, the field width and break mask will be returned to the user at BLOCK+3.

IO.ABS mode allows you to control the field width using the .TOSBS function. The mode can be set or cleared at any time. Clearing the mode clears the break mask, and setting the mode creates a new table of break masks. The default table has no break characters and a field width of one, thus setting the terminal up for single-character I/O.

Even when IO.ABS is set, echoing is controlled by IO.SUP. If IO.SUP is set in the I/O status word, no characters are echoed by the monitor. If IO.SUP is not set, printing characters (ASCII codes 1Ø1 to 176) are echoed by the monitor, and control characters that are not defined as break characters are echoed. Control characters defined as break characters will never be echoed. This includes RETURN, line feed, vertical tab, and bell characters.

When IO.ABS is set, the monitor will do no line editing functions, so CTRL/T, CTRL/W, CTRL/U, CTRL/R, and DELETE will not function. IO.ABS mode has no effect on CTRL/C processing, however.

Setting IO.ABS also implies deferred echo mode. Characters are not echoed until the program requests input from the terminal. The monitor postpones the echo of any characters after a break character is typed, until the program attempts to read the next field. That field is echoed up to its break character, and so forth. This prevents echoing during any change in IO.SUP or IO.ABS states, and ensures synchronization of echo and program output.

15.5 TERMINAL I/O

TTCALLs operate only on a controlling terminal. TRMOP. must be used for other terminals and may also be used for the controlling terminal.

In addition to its general I/O monitor calls, TOPS-10 offers a number of special terminal monitor calls to make terminal I/O simpler. Special terminal monitor calls include:

GETLIN	Returns the SIXBIT physical name of the controlling terminal.
TRMNO.	Returns the Universal Device Index for the controlling terminal. This index includes the terminal number in bits 27-35.
TRMOP.	Performs many useful terminal operations. See the TRMOP. functions in Chapter 22.
INCHRW	Inputs a character from the controlling terminal (waits for the character).
OUTCHR	Outputs a character to the controlling terminal.
INCHRS	Inputs a character from the controlling terminal (skips only if a character is received).
OUTSTR	Outputs an ASCIZ string to the controlling terminal.
INCHWL	Inputs a character from the controlling terminal only if a break character has been typed.
INCHSL	Inputs a character from the controlling terminal only if a break character has been typed and skips if the character is received.
GETLCH	Returns the characteristics for a terminal line.
SETLCH	Sets the characteristics for the controlling terminal.
RESCAN	Allows your program to read the monitor command that invoked it.
CLRBFI	Clears any typeahead on the controlling terminal. Useful following detection of an error on a previous command.
CLRBFO	Clears the buffer for output to terminal.

SKPINC Skips if at least one character has been typed on the controlling terminal.

SKPINL Skips if at least one line has been typed on the controlling terminal.

IONEOU Displays an 8-bit image mode character on the controlling terminal.

CHTRN. Translates characters from one interpretation to another. For instance, you can use CHTRN. to translate 8-bit characters to 7-bit characters. You should use this UUO instead of writing a program for character translation, because CHTRN. translates using an ANSI standard table.

15.6 NON-BLOCKING TERMINAL I/O

A program can perform non-blocking I/O with the terminal by posting a read request for the terminal. You can do this using either of the following methods:

- A TTCALL, such as:
  a. SKPINL to select line mode without committing to actually reading a character or line.
  b. INCHRS or INCHSL to select the mode and to commit to accepting any available character or line.
- A HIBER call, setting HB.RTC to set character mode, or HB.RTL to set line mode. In this case, the program will HIBER until a WAKE is made, and the program must determine the reason for awakening.

Otherwise, the program can rely on the PSI system (described in Chapter 6) to interrupt the program when a character or line is ready to be processed.

Your program must use one of the above methods for non-blocking I/O if the terminal is set to deferred echo mode. In deferred echo mode, the monitor echoes no characters until the program has posted a read request, and the terminal input cannot be processed until the characters have been echoed. The program need not block while waiting for input, but it must indicate its ability to accept commands by using one of the above methods, so that the monitor will echo terminal input at the first opportunity.

The user program must be aware of the fact that input character echoing activity has a lower priority to the monitor than output character activity from the program. As long as the program is sending characters to the terminal, all input characters are being stored by the monitor, waiting for an opportunity to echo to the terminal, in deferred echo mode. It is possible for a program to keep the terminal busy processing output, so that the terminal will never be available for input. The program must allow for this possibility by stopping output or by checking the terminal input buffer to determine whether echoing is required. Use the TRMOP. function .TOECC to determine whether echoing is required.

15.7 TERMINAL PAPERTAPE I/O

If a terminal has a papertape device attached, it can perform papertape I/O. To enable papertape I/O from the terminal, issue the SET TTY TAPE monitor command. This enables the four required characters: CTRL/Q (XON), CTRL/S (XOFF), CTRL/R (AUX ON), and CTRL/T (AUX OFF).

15.7.1 Using Terminal Papertape Input

To begin reading from a terminal papertape, you must type CTRL/Q. Papertape input ends when a CTRL/S is read from the papertape or from the terminal keyboard.

The following example shows how to read from a terminal papertape reader into the disk file DSK:FILE.FIL:

```
.SET TTY TAPE
.R PIP
*DSK:FILE.FIL=TTY:
^QThis line is from the papertape.
So is this one.
And even this last one.
^Z
^S^C
.
```

15.7.2 Using Terminal Papertape Output

The terminal papertape punch is connected in parallel with the keyboard printer; therefore when the punch is on, all characters typed on the keyboard are punched on the papertape.

Papertape output begins with a CTRL/R typed at the terminal; for the LT33B or LT33H terminal, a CTRL/R character output from a program can begin papertape output.

Papertape output continues until a CTRL/T character occurs in the punched output (either from a program or from the terminal keyboard). The CTRL/T is the last character punched on the papertape.

If your program is punching papertape at a terminal, it should punch several inches of blank tape before the final CTRL/T; this permits you to tear off and discard the CTRL/T character at the end of the tape.

15.8 TERMINAL I/O STATUS

The I/O status bits for the terminal are listed below. The error flags are set by the monitor, but the I/O status bits (bits 23 through 35) can be set by your program to initiate different modes of I/O.

Bits	Symbol	Meaning
18-21	IO.ERR	Error flags. These are returned by the monitor after an I/O operation that failed:

Flag	Symbol	Error
18	IO.IMP	Not assigned to a job for image mode input.
19	IO.DER	Ignore interrupts for Ø.75 seconds.
2Ø	IO.DTE	Echo failure on output.
21	IO.BKT	Character lost on input.

If all of these bits are set, the remote node to which the terminal is connected has failed.

23	IO.ACT	Device active.
25	IO.ABS	Enable using break mask. (Refer to Section 15.4.)
26	IO.BKA	Break on all characters. An IN or INPUT call will terminate on the first character typed, thus enabling character input mode.
27	IO.TEC	Truth in echoing mode. This causes all control characters to be output as themselves (for example, the ESCAPE character is not echoed as $ but as octal 33).
28	IO.SUP	Suppress echoing.
29	IO.LEM	Enable special editor mode. This causes some control characters (CTRL/R, CTRL/H, CTRL/U, CTRL/W, and DELETE) to be passed to the program and ignored by the monitor, except to echo the control characters and to act as break characters.
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
Ø	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
2	.IOPIM	Packed image mode.
4	.IOAS8	8-bit ASCII mode.
1Ø	.IOIMG	Image mode.

Using the TRMOP. call's function .TOBKA, your program can test whether the terminal is in line input or character input mode. Unless you set the file status bit IO.BKA, line input mode is assumed.

15.9 PSEUDO-TERMINALS

Most jobs that run on the TOPS-1Ø monitor are initiated by a user at a terminal with the LOGIN command. Unless the job is detached by the DETACH command, it remains under control of the terminal until is is logged out (using KJOB).

A program can control a job by simulating a terminal. The pseudo-terminal (a software-defined device) simulates terminal interaction between the job and the pseudo-terminal (called a PTY). The controlling program (for example, the batch processor) uses the PTY device in the same way that a user uses a physical terminal. It initiates the job on the PTY, provides input, accepts output, and closes the PTY, all by means of monitor calls.

A controlling job (BATCON, for example) communicates with controlled jobs (such as batch jobs) through a PTY. The PTY device handles the terminal commands for the controlling job. It passes input to the terminal input buffer of the controlled job and passes output from the controlled job's terminal output buffer to the controlling job. Input is only echoed if the full-SCNSER PTY bit is set. Figure 15-1 illustrates PTY I/O.

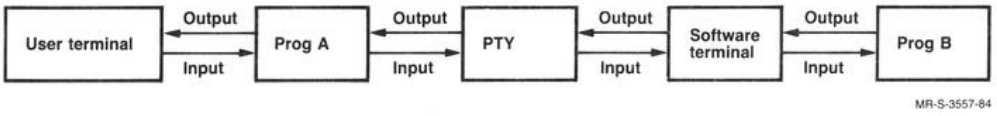


Figure 15-1: PTY I/O

In Figure 15-1, Program A is the controlling job. For example, Program A might be OPSER. Program B is the controlled job. Program B performs I/O just as though it were performing I/O with a command terminal, by placing I/O into its terminal I/O buffer. However, the PTY replaces the controlling terminal function by passing I/O to and from the controlling job (Program A).

A PTY is never in I/O wait state. Therefore, a program will not block for PTY I/O. This allows the program to control multiple PTYs. Therefore, the program can use HIBER, read I/O status bits, or JOBSTS to determine whether I/O from a PTY is necessary.

The buffer size for a PTY is the same as for a terminal: 23 (octal) words.

15.9.1 Pseudo-Terminal Names

The device name of a PTY is of the form:

PTYnnn

Where nnn is a 3-digit octal number (leading zeros must be omitted).

15.9.2 Pseudo-Terminal I/O

A number of monitor calls are especially important to a PTY-controlled program. These include:

OPEN — Sets up for PTY I/O. Optionally, you may set the full-SCNSER PTY bit, allowing the PTY to be controlled directly by a terminal. Setting this bit creates a PTY with all the characteristics of a terminal. (Some of the TRMOP. functions are meaningless for a PTY.)

HIBER — Allows the controlling program to temporarily suspend execution until either of the following occurs:

- The PTY has I/O. If your controlling program set HB.RPT when is used the HIBER call, your program will be woken whenever I/O is necessary.
- A specified amount of time has passed since your program went to sleep. Your program will wake even if no activity has occurred in the controlled job. Your program should check JOBSTS to determine whether the PTY has I/O. Your controlling program can abort any controlled programs by sending two CTRL/Cs to the PTY of the controlled job.

If the controlling job need not interrupt the controlled jobs, it should put itself back to sleep with another HIBER call.

If your program does not set the HB.RPT bit for a HIBER call, the monitor wakes the controlling job every time there is a change in the I/O status bits. This prevents your job from sleeping when it should be servicing the controlled jobs.

OUT/OUTPUT

The output is sent from your controlling program to the PTY, and thus read by the controlled job from its terminal input buffer, just as though it came from its own terminal. When your program performs an OUT or OUTPUT to the controlled job, the first OUT/OUTPUT after an OPEN, INIT, or FILOP. causes the monitor to discard the contents of the PTY's output buffer. (Refer to the RELEAS monitor call, below.) Thus, your program should send a "dummy" OUT/OUTPUT before its first real output.

The OUT or OUTPUT calls cause the monitor to:

1. Place characters from your controlling program's buffer ring into the input buffer of the terminal linked to the PTY.
2. Clear the IO.PTI bit in the I/O status word, so that the job is not in input wait state.
3. Set or clear the IO.PTM bit in the I/O status word, depending on the state of the terminal.

For PTY output, the monitor also:

1. Discards all null characters (ASCII ØØØ).

2. If more output is performed than the PTY can accept, the monitor sets the IO.BKT bit in the I/O status word, discarding the remainder of the controlling program's output buffer.

3. The monitor also translates all lowercase characters sent to the PTY to uppercase, if the appropriate bit is not set for the PTY, using TRMOP. function .TOLCT. Many of the TRMOP. characteristics can be set for a full-SCNSER PTY. (See the OPEN call in Chapter 22.)

IN/INPUT

The input comes from the controlled job, which places the data in the PTY's output buffer, from which your controlling job can read it with IN or INPUT. If no characters are read, the monitor returns an empty buffer. An INPUT call does not cause a wait state.

The monitor passes all available characters to your controlling program. If there are more characters than can fit in the buffer of the controlling program, the monitor sets bit IO.PTO in the file status word and waits for another INPUT monitor call. When the terminal's buffer is emptied by an INPUT call, the monitor clears bit IO.PTO.

RELEAS For a PTY, the RELEAS call causes the monitor to:

1. Discard the contents of the terminal's output buffer.

2. Detach the controlled job from the terminal (if it is attached).

3. Release the PTY from its channel.

NOTE

Haphazard use of RELEAS with PTYs may leave detached jobs on the system; these use system resources unnecessarily.

JOBSTS Returns data about the PTY and the controlled job.

CTLJOB On a normal (skip) return, this call returns the job number of the controlling job; you must specify the job number of the controlled job. On an error return, you specified an invalid (non-existent) job number. The AC is not cleared on an error return.

15.10 PSEUDO-TERMINAL DATA MODES

A pseudo-terminal can use ASCII or ASCII line mode. These modes are identical in action to the same modes for terminal devices.

15.11 PSEUDO-TERMINAL I/O STATUS

The I/O status bits for the PTY are:

Bits	Symbol	Meaning
21	IO.BKT	More output sent than was accepted by the PTY.
23	IO.ACT	Device active.
24	IO.PTI	Job is waiting to receive input.
25	IO.PTO	Job is waiting to send output.
26	IO.PTM	Subjob is in monitor mode.
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
Ø	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
2	.IOPIM	Packed image mode (for full-SCNSER PTYs only).
4	.IOAS8	8-bit ASCII mode.

CHAPTER 16

LINE PRINTERS (LPT)

TOPS-10 supports up to three line printers for each CPU in a KL system. For KS systems, TOPS-10 supports one line printer.

16.1 LINE PRINTER NAMES

The physical names of the line printers are LPTnn0, LPTnn1, and LPTnn2, where nn is the node number for the system.

16.1.1 Controller Names

The controller name for the line printers is either BA10, LP100, LP200, LP11, or LP20.

16.1.2 Unit Names

The unit name of a line printer is LSP10-LA, LSP10-LB, LP10-Fc, LP10-Hc, (where c is A, B, C, or D), LPT20xx, LP14, LP25, LP26, LP27, LP05, LP14, LP07, or LN01. The LN01 is a laser printer.

16.2 LINE PRINTER DATA MODES

The buffer size for a line printer is 37 (octal) words.

A line printer can use any of the following data modes:

- In ASCII mode, ASCII characters are sent to the line printer exactly as they appear in the buffer. The line printer prints from 1 to 8 spaces for a tab, feeds four lines for a vertical tab, and skips to top-of-page for a formfeed; a linefeed moves the paper up one line and returns the carriage to the left position; a carriage-return returns the carriage return to the left position.

- ASCII line mode is identical to ASCII mode.

- 8-bit ASCII mode is allowed only on a line printer attached to a network line. Otherwise, it functions identically to ASCII mode.

- Image mode is identical to ASCII mode.

16.3 LINE PRINTER I/O

The monitor normally sends a formfeed to the line printer on the first OUT or OUTPUT call, and on a CLOSE call. Your program can suppress these formfeeds by setting the IO.SFF bit in the I/O status word.

The DEVOP. call reads and sets various line printer characteristics, and performs other operations for line printer. For local-line printers and console front-end printers, use DEVSTS. to read line printer characteristics. Refer to Chapter 22 in Volume 2.

16.4 LINE PRINTER I/O STATUS

The I/O status bits for the line printer are as follows:

Bits	Symbol	Meaning
23	IO.ACT	Device active.
25	IO.SVF	Suppresses the vertical format unit on a line printer. This allows LNØ1 fonts and graphics to print correctly. (Refer to the TOPS-1Ø Operating System Commands Manual for information on using LNØ1 fonts).
29	IO.SFF	Suppress formfeeds on OUT, OUTPUT, and CLOSE monitor calls.
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
Ø	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
4	.IOAS8	8-bit ASCII mode, for network line printers only.
1Ø	.IOIMG	Image mode.

CHAPTER 17

CARD READERS (CDR) AND CARD PUNCHES (CDP)

TOPS-10 supports two card readers and 2 card punch units for each CPU (KL processors only) on the system. For KS processors, one reader is supported, but no card punch device is supported. A card reader is an input device that can be connected to an MPX (multi-plexed) channel. A card punch is an output device to punch data on punched cards, which are then read by the card reader.

A card reader can read cards in either 026 code or in ANSI code. The first card contains a punch code in column 1 that shows which code is used: 12-11-8-9 for 026 code or 12-0-2-4-6-8 for ANSI code.

See the Commands Manual for a table of ASCII codes, ANSI card codes, 026 card codes, and binary coded decimal (BCD) codes.

The end-of-file for a card deck is indicated in one of two ways:

- The deck ends with an end-of-file card (12-11-0-1-6-7-8-9 in column 1).
- The operator or user presses the end-of-file key on the card reader.

17.1 CARD DEVICE NAMES

The card reader names are CDRnn0 and CDRnn1 and the card punch names are CDPnn0 and CDPnn1, where nn is the node number.

The unit name of a card reader is one of the following: CR10-D, CR10-E, or CR10-F.

The unit name of a card punch is either CP10A or CP10D.

17.2 CARD READER DATA MODES

A card reader can use any of five data modes:

1. In ASCII mode,all 80 columns of each card are translated to ASCII characters and placed in the buffer. A header card can be the first card in the file; if so, this card indicates in column 1 whether the deck is in 026 code (12-11-8-9 punch) or in ANSI code (12-0-2-4-6-8 punch).

   The card reader service routine inserts a carriage-return and a linefeed after each card. As many cards as possible are read into the buffer; data from a card is not divided between buffers. For the default buffer size of 24 (octal) words, data from only one card is placed in a buffer.

2. ASCII line mode is identical to ASCII mode.

3. In image mode, each buffer is 36 (octal) words and receives one card of 80 characters. Each character is placed in the buffer as a 12-bit byte; the last byte of the buffer is not used.

4. In image binary mode, column 1 of each card must contain a 7-9 punch (to verify that the mode is image binary). If this punch is missing, the monitor sets the IO.IMP bit in the file status word. Column 1 also contains the word count for the card in punch fields 12 through 3. Column 2 contains a 12-bit folded checksum. For image binary mode, a buffer is 35 (octal) words long.

   The folded checksum is formed by adding the data words (two's complement arithmetic), then dividing the result into three 12-bit bytes that are added (one's complement arithmetic).

5. In superimage mode, the 36 bits from the I/O bus are placed (BLKI) directly into the buffer. To use superimage mode, your program must set the IO.SIM bit in the file status word. The buffer size in superimage mode is 123 (octal) words.

17.3 CARD PUNCH DATA MODES

A card punch can use any of five data modes:

1. In ASCII mode, characters are read from the buffer and punched in a card code (026 code if IO.D29 is not set; ANSI code if IO.D29 is set). Each buffer contains up to 80 ASCII characters, and is punched on one card. Tabs are simulated by punching from 1 to 8 blanks; a linefeed ends a card and begins a new one; formfeeds and carriage-returns are ignored; all other nontranslatable characters are punched as question marks. A buffer in ASCII mode is 23 (octal) words.

   A card can be split between buffers, but an attempt to punch more than 80 characters on a card sets the IO.BKT bit in the file status word.

On the first OUT or OUTPUT call to the card punch, an entire card is punched indicating which card code is used: 12-2-4-8 punches for Ø26 code; 12-2-4-6-8 punches for ANSI code.

On a CLOSE monitor call to the card punch, the monitor punches the last card (from the buffer) and punches an end-of-file card.

2. ASCII line mode is identical to ASCII mode.

3. In image mode, each 12-bit byte of data from the buffer is punched as one card column; the last byte in the last word is discarded. (The monitor usually sets up this handling using 36-bit bytes; if your program specifies 12-bit bytes, it must skip the last byte in the buffer.) For image mode, the buffer size is 36 (octal).

   On a CLOSE monitor call to the card punch, the monitor punches the last card (from the buffer) and punches an end-of-file card.

4. In image binary mode, one card is punched for each output buffer. A buffer is 36 (octal) words. Your program should not force output after each 8Ø columns, since this wastes disk space if the output file is spooled.

   On a CLOSE monitor call to the card punch, the monitor punches the last card (from the buffer) and punches an end-of-file card.

5. In binary mode, each card contains data in columns 3 through 8Ø. A buffer is 35 (octal) words. Column 1 contains a binary word count in punches 12 through 3, and a 7-9 punch; column 2 contains a folded checksum.

   The folded checksum is formed by adding the data words (two's complement arithmetic), then dividing the result into three 12-bit bytes that are added (one's complement arithmetic).

17.4 CARD DEVICE I/O

On each interrupt for card readers and punches connected to the I/O bus, the device status word is updated (it stores the result of a CONI in the DDB); use the DEVSTS monitor call to retrieve the status.

On a CLOSE monitor call to the card punch, the monitor punches the last card (from the buffer) and punches an end-of-file card. The end-of-file card and columns 2 through 8Ø of the header card contain the same punch that appears in column 1 for file identification. On input, these punches are ignored by the device service routine.

17.5 CARD DEVICE I/O STATUS

The I/O status bits for the card reader and card punch are as follows:

Bits	Symbol	Meaning
18-21	IO.ERR	Error flags:

Flag	Symbol	Error
18	IO.IMP	Column 1 of a card presumed binary did not have 7-9 punch; the reader is stopped. (For card reader only.)
19	IO.DER	For card reader, a photocell error occurred, indicating that a card motion error caused data to be missed; the reader is stopped. For card punch, a punch error occurred.
20	IO.DTE	Checksum read from card different from computed checksum; the reader is stopped. (For card reader only.)
21	IO.BKT	End-of-file reached with data still in buffer. Attempted to punch more than 80 columns on one card (card punch only).

22	IO.EOF	End-of-file card read or end-of-file key pressed (card reader only).
23	IO.ACT	Device active.
29	IO.SIM	Superimage mode for card readers.
29	IO.D29	For card punches, specifies that ANSI codes should be punched in ASCII mode. If this bit is cleared, punch codes will be 026.
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
0	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
10	.IOIMG	Image mode.
13	.IOIBN	Image binary mode.
14	.IOBIN	Binary mode.

CHAPTER 18

PAPERTAPE READERS (PTR) AND PUNCHES (PTP)

TOPS-1Ø supports one papertape reader and one papertape punch for each CPU on the system.

18.1 PAPERTAPE DEVICE NAMES

The physical name of the papertape reader on a system is PTRnnØ and the name of a papertape punch is PTPnnØ, where nn is the node number of the system for the reader.

The unit name of a papertape reader is PCØ4 or PCØ5. The unit name of a papertape punch is PCØ9.

18.2 PAPERTAPE READER DATA MODES

The buffer size for a papertape reader is 43 (octal) words.

A papertape reader uses any of five data modes: ASCII, ASCII line, image, binary image, or binary. For all these data modes, the physical end-of-tape sets the end-of-file bit (IO.EOF) in the file status word, but does not send an end-of-file character to the buffer.

1. In ASCII mode, the monitor ignores blanks (ØØØ), delete characters (377), and nulls (2ØØ). The parity bit (punch 8) is not sent to the buffer, so that characters put into the buffer are 7-bit ASCII.

2. In ASCII line mode, the monitor behaves the same as in ASCII mode, except that each line ends with a linefeed, a formfeed, or a vertical tab.

3. In image mode, 8-bit characters are copied into the buffer exactly as they are received from the reader.

4. In image binary mode, if punch 8 is not punched, the character is ignored; otherwise, the first six punch fields are sent to the buffer as a SIXBIT character.

5. In binary mode, the monitor reads checksummed binary data into the buffer. The right half of the first word of each physical block contains the number of data words that follow; the left half contains half of the folded checksum.

   The folded checksum is formed by adding the data words (two's complement arithmetic), then dividing the result into three 12-bit bytes that are added (one's complement arithmetic).

   The maximum block length is 40 (octal) words. The byte pointer is initialized to point to the second word, skipping the word containing the word count and folded checksum.

18.3 PAPERTAPE PUNCH DATA MODES

The buffer size for a papertape punch is 43 (octal) words. A papertape punch uses any of five data modes:

1. In ASCII mode, ASCII characters are sent to the punch. For each character, the 8th hole is punched if needed to make even parity.

2. In ASCII line mode, ASCII characters are sent to the punch. A tapefeed character (000) is inserted after each formfeed. A delete character is inserted after each vertical tab and horizontal tab. Nulls are deleted. ASCII line mode is identical to ASCII mode.

3. In image mode, 8-bit characters are sent to the punch, exactly as they appear in the buffer.

4. In image binary mode, SIXBIT characters are sent to the punch, exactly as they appear in the buffer. For each character, the 7th hole is left unpunched and the 8th is punched. There is no format control, and no checksumming.

5. In binary mode, each bufferful of data is sent to the punch as a single checksummed binary block. The format of this block is described in the previous section. Several blank characters are punched after each bufferful to provide visual clarity.

18.4 PAPERTAPE I/O

On each interrupt, the papertape device status word is updated (it stores the result of a CONI in the DDB); use the DEVSTS monitor call to retrieve the status.

On the first OUT or OUTPUT call to the papertape punch, the monitor sends two fanfolds of blank tape to the punch; on a CLOSE call, the monitor sends one fanfold of blank tape. A CLOSE call does not automatically append an end-of-file character to the data sent.

18.5 PAPERTAPE I/O STATUS

The I/O status bits for the papertape devices are listed below. Note that IO.ERR (bits 18-21) can be set only for papertape readers.

Bits	Symbol	Meaning
18	IO.IMP	Incomplete binary block.
20	IO.DTE	Bad checksum in binary mode.
22	IO.EOF	Physical end-of-tape found; no end-of-file character is in the buffer.
23	IO.ACT	Device active (for readers and punches).
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
0	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
10	.IOIMG	Image mode.
13	.IOIBN	Image binary mode.
14	.IOBIN	Binary mode.

CHAPTER 19

PLOTTERS (PLT)

TOPS-10 supports two plotters for each CPU on the system (KL processor only).

19.1 PLOTTER DEVICE NAMES

The physical names of plotters are PLTnn0 and PLTnn1, where nn is the node name.

19.1.1 Controller Names

The controller name for the plotters is XY10.

19.1.2 Unit Names

The unit names for the plotters are XY10A and XY10B.

19.2 PLOTTER DATA MODES

Data modes for the plotter are ASCII, ASCII line, image, image binary, and binary.

For ASCII and ASCII line modes, five 7-bit characters are transmitted per word. The monitor drops the leftmost bit of each 7-bit ASCII character to form a SIXBIT character.

For image, binary image, and binary modes, the monitor sends the contents of the buffer without change. Six SIXBIT characters are transmitted per word.

19.3 PLOTTER I/O

The buffer size for a plotter is 46 (octal) words. This buffer contains characters that are interpreted by the plotter service routine in sets of six SIXBIT bytes, as follows:

1. First in set: Raise pen.
2. Second in set: Lower pen.
3. Third in set: Move drum up (-x).
4. Fourth in set: Move drum down (x).
5. Fifth in set: Move carriage left (-y).
6. Sixth in set: Move carriage right (y).

Your program cannot combine pen movements with drum or carriage movements. See the Hardware Reference Manual.

Some monitor calls have special behavior for the plotter:

1. On the first OUT or OUTPUT call, a raise-pen command is prefixed to the output.
2. On a CLOSE call, a raise-pen command is prefixed to the data remaining in the buffer.
3. After each interrupt, plotter status is updated (it stores the result of a CONI in the DDB); to retrieve the status, use the DEVSTS call.

19.4 PLOTTER I/O STATUS

The I/O status bits for the plotter are as follows:

Bits	Symbol	Meaning
23	IO.ACT	Device active.
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
Ø	.IOASC	ASCII mode.
1	.IOASL	ASCII line mode.
1Ø	.IOIMG	Image mode.
13	.IOIBN	Image binary mode.
14	.IOBIN	Binary mode.

CHAPTER 20

DISPLAY LIGHT PENS (DIS)

The device service routine for a display light pen device guarantees a flicker-free picture if your job is locked in core; if the system is lightly loaded, the picture may be satisfactory even if your job is not locked in core. To lock your job in core, refer to the LOCK call.

20.1 DISPLAY LIGHT PEN NAMES

The physical name of the display light pen is DIS.

20.1.1 Unit Names

The TOPS-10 monitor supports the following display light pen units: VR30, VB10C, Type 30, and type 340.

20.2 DISPLAY LIGHT PEN DATA MODES

A display light pen uses only image dump mode for its I/O; the device uses no buffer.

20.3 DISPLAY LIGHT PEN I/O

For display light pen I/O, a few monitor calls behave in special ways. The data mode for the device must be image dump mode (.IOIDP in IO.MOD). The number of buffers must be zero.

On an IN or INPUT call, the value returned at the specified address is the location of the last light pen hit, or -1 if none was detected.

On an OUT or OUTPUT call, the address given in the call is the address of a table of commands. These commands are of four types:

1. An output command of the form:

   IOWD buflength,buffer

   where buflength is the length of the output buffer; and buffer is the address of the buffer.

2. A skip command of the form:

   XWD Ø,nextcmd

   where **nextcmd** is the address of the next command to be read.

3. An intensity command (except for Type 34Ø) of the form:

   XWD intensity,Ø

   where **intensity** is the value of the desired intensity. These values range from 4 (dim) to 13 (bright).

4. An end-list command of the form:

   Z

   that ends the command list.

The following two examples show how to use these command lists for display pen devices. The first example is for a VR3Ø device. The values at the last six labels define data for the device.

```
        OUTPUT  CHANNO,LIST         ;Output data at command list
        . . .
LIST:   XWD     5,Ø                 ;Intensity 5 (DIM)
        IOWD    1,A                 ;Plot coordinates A
        IOWD    5,SUBP1             ;Plot subpicture SUBP1
        XWD     13,Ø                ;Intensity 13 (BRIGHT)
        IOWD    1,C                 ;Plot coordinates C
        IOWD    2,SUBP2             ;Plot subpicture SUBP2
        XWD     Ø,LIST1             ;Skip to LIST1 for next command
        . . .
LIST1:  XWD     1Ø,Ø                ;Intensity 1Ø (NORMAL)
        IOWD    1,B                 ;Plot coordinates B
        IOWD    1,D                 ;Plot coordinates D
        Z                           ;End command list

A:      XWD     6,6                 ;X=6      Y=6
B:      XWD     1Ø5,7Ø              ;X=1Ø5    Y=7Ø
C:      XWD     7Ø,1Ø5              ;X=7Ø     Y=1Ø5
D:      XWD     1ØØØ,2ØØ            ;X=1ØØØ   Y=2ØØ
SUBP1:  BLOCK   5                   ;Subpicture 1
SUBP2:  BLOCK   2                   ;Subpicture 2
```

The next example shows how to use the command list for a Type 340 display light pen. In this example, the coordinates and subpictures are used more than once.

```
        OUTPUT  CHANNO,LIST         ;Output data at command list
        . . .
LIST:   IOWD    1,A                 ;Start at (6,6)
        IOWD    5,SUBP1             ;Draw a circle
        IOWD    1,C                 ;Set to (70,105)
        IOWD    5,SUBP1             ;Draw another circle
        IOWD    1,B                 ;Set to (105,70)
        IOWD    2,SUBP2             ;Draw a triangle
        XWD     0,LIST1             ;Skip to LIST1 for next command
        . . .
LIST1:  IOWD    1,D                 ;Set to (1000,200)
        IOWD    5,SUBP1             ;Draw a circle
        IOWD    1,A                 ;Set to (6,6)
        IOWD    2,SUBP2             ;Draw a triangle
        Z                           ;End of command list
A:      XWD     6,6                 ;X=6      Y=6
B:      XWD     105,70              ;X=105    Y=70
C:      XWD     70,105              ;X=70     Y=105
D:      XWD     1000,200            ;X=1000   Y=200
SUBP1:  BLOCK   5                   ;A circle
SUBP2:  BLOCK   2                   ;A triangle
```

20.4 DISPLAY I/O STATUS

The I/O status bits for the display are as follows:

Bits	Symbol	Meaning
23	IO.ACT	Device active.
32-35	IO.MOD	Data mode:

Code	Symbol	Meaning
15	.IOIDP	Image dump mode.

CHAPTER 21

REMOTE DATA TERMINALS (RDA)

A remote data terminal is a multidrop or intelligent buffered terminal. Remote data terminal devices exist only on DN8Ø-series remote concentrators running network software (ANF-1Ø). The monitor sets bit 35 in the DEVSTS word if the device is a multidrop device.

A remote data terminal can be multiplexed with other devices on an MPX channel. It can perform nonblocking I/O and can be used for programmed interrupts (refer to Chapter 6).

21.1 REMOTE DATA TERMINAL NAMES

A remote data terminal is identified by a device name in the form:

RDcnnu

where c is a letter in the range A through H; nn is a 2-digit octal node number in the range 1 to 77; and u is a 1-digit octal unit number in the range Ø to 7.

For example, the device name RDAØ13 identifies the remote data terminal on controller A at node 13 with unit number 3.

NOTE

No generic searches are allowed or performed for remote data terminals; only the specific device name can be used.

21.2 REMOTE DATA TERMINAL I/O

A remote data terminal uses a buffer of 1Ø3 (octal) words; three of these are buffer header words. The first word contains a 2-character function code (in ASCII) in bits Ø to 14; and (for multi-drop lines) a 3-character right-justified multidrop number in bits 15 to 35. The multidrop number must have leading ASCII blank or zeros as needed.

21.3 REMOTE DATA TERMINAL DATA MODES

The monitor performs no processing of the data for remote data terminals. The device sends and receives data exactly as it appears in the buffer. The monitor does not insert fillers or delete characters that are followed by RUBOUTs.

21.4 REMOTE DATA TERMINAL I/O STATUS

The I/O status bits for the remote data terminal are as follows:

Bits	Symbol	Meaning
18-21	IO.ERR	Error flags:

Flag	Symbol	Error
18	IO.IMP	Line number not in polling sequence. This bit can be set on an OUT/OUTPUT error, indicating that an illegal drop number was specified. (Drop number less than 5 characters or contains non-alphanumeric characters.)
19	IO.DER	Terminal is in polling list, but device failed. This bit can be set on either IN/INPUT or OUT/OUTPUT, if the network connection is lost for some reason. The DDB is not deleted, but is retained, with the device name set to ___nnu, rather than RDxnnu. This is the name assigned to unknown devices until the user releases the device.
2Ø	IO.DTE	Error on the entire multi-drop line.
21	IO.BKT	User buffer exceeded maximum length of DDCMP message. This bit is set when the message is too large for the I/O buffer.

22	IO.EOF	End-of-file reached.
23	IO.ACT	The device is active.

INDEX