PDP-10 REAL-TIME SOFTWARE

Real-Time Calls
Monitor Algorithms

digital
Digital Equipment Corporation, Maynard, Mass. 01754

The material in this manual is for information only and is subject to change without notice.

The following are trademarks of Digital Equipment Corporation, Maynard, Massachusetts:

DEC	PDP
FLIPCHIP	FOCAL
DIGITAL	COMPUTER LAB

FOREWORD

The material in this publication was taken from PDP-10 TOPS Monitor Calls (DEC-10-MRRA-D). Anyone interested in the real-time capabilities of TOPS-10 will find these excerpts helpful.

Other relevant publications include:

TOPS-10 Operating System Commands (DEC-10-MRDA-D), available fall, 1971

PDP-10 Timesharing Handbook

PDP-10 Reference Handbook

PDP-10 Computer Networks

PDP-10 Applications in Science

TABLE OF CONTENTS

CONVENTIONS USED IN TOPS-10 MONITOR CALLS

The following conventions have been used throughout this manual:

dev:	Any logical or physical device name. The colon must be included when a device is used as part of a file specification.
list	A single file specification or a string of file specifications. A file specification consists of a filename (with or without a filename extension), a device name if the file is not on disk, a project-programmer number, if the file is not in the user's disk area, and a protection.
arg	A pair of file specifications or a string of pairs of file specifications.
job n	A job number assigned by the monitor.
file.ext	Any legal filename and filename extension.
core	Decimal number of 1K blocks of core.
adr	An octal address.
C(adr)	The contents of an octal address.
[proj,prog]	Project-programmer numbers; the square brackets must be included in the command string.
fs	Any legal file structure name or abbreviation.
(\$)	The symbol used to indicate an altmode.
↑x	A control character obtained by depressing the CTRL key and then the character key x.
←	A back arrow used in command string to separate the input and output file specifications.
*	The system program response to a command string.
.	The monitor response to a command string.
	The symbol used to indicate that the user should depress the RETURN key. This key must be used to terminate every command to the monitor command language interpreter.
———	Underscoring used to indicate computer typeout.
n	A decimal number.

3.1.4.2 HIBER AC, or CALLI AC, 72

The HIBERNATE UUO allows a job to become dormant until a specified event occurs. The possible events that can wake a hibernating job are: 1) input activity from the user's TTY or any TTY INITed by this job (both line mode and character mode), 2) PTY activity for any PTY currently INITed by this job, 3) the time-out of a specified amount of sleep time, or 4) the issuance of a WAKE UUO directed at this job either by some other job with wake-up rights or by this job at interrupt level.

The HIBERNATE UUO must contain in the left half of AC the wake-condition enable bits and in the right half the number of ms for which the job is to sleep before it is awakened.

The call is as follows:

```
MOVSI AC, enable bits      ;get HIBERNATE conditions
HRRI AC, sleep time        ;number of ms to sleep
HIBER AC,                  ;or CALLI AC, 72
error return
normal return
```

The HIBERNATE UUO enable condition codes are as follows:

Bits	Meaning
18-35	Number of ms sleep time. Is rounded up to an even multiple of jiffies (maximum being 2^{12} jiffies). Zero means no clock request (i.e., infinite sleep).
15-17	WAKE UUO protection code: Bit 17 = 1, project codes must match. Bit 16 = 1, programmer codes must match. Bit 15 = 1, only this job can wake itself.
13-14	Wake on TTY input activity: Bit 14 = 1, wake on character ready. Bit 13 = 1, wake on line of input ready.
12	Wake on PTY activity since last HIBERNATE.

An error return is given if the UUO is not implemented. The SLEEP UUO should be used in this case. A normal return is given after an enabled condition occurs.

Jobs either logged-in as [1,2] or running with the JACCT bit on can wake any hibernating job regardless of the protection code. This allows privileged programs, which are the only jobs that can wake certain system jobs, to be written.

A RESET UUO always clears the protection code and wake-enable bits for the job. Therefore, until the first HIBERNATE UUO is called, there is no protection against wake-up commands from other jobs. To guarantee that no other job wakes the job, a WAKE UUO followed by a HIBERNATE UUO with

the desired protection code should be executed. The WAKE UUO ensures that the first HIBERNATE UUO always returns immediately, leaving the job with the correct protection code.

3.1.4.3 WAKE AC, or CALLI AC, 73 - The WAKE UUO allows one job to activate a dormant job when some event occurs. This feature can be used with Batch so that when a job wants a core dump taken, it can wake up a dump program. Also, real-time process control jobs can cause other process control jobs to run in response to a specific alarm condition. The WAKE UUO can be called for a RTTRP job running at interrupt level (refer to Paragraph 3.8.1), thereby allowing a real-time job to wake its background portion quickly in order to respond to some real-time condition.

The call is as follows:

```
MOVE AC, JOBNUM          ;number of job to be awakened
WAKE AC,                 ;or CALLI AC, 73
error return
normal return
```

An error return is given if the proper wake privileges are not specified. There is a wake bit associated with each job. If any of the enabled conditions specified in the last HIBERNATE UUO occurs, then the wake bit is set. The next time a HIBERNATE UUO is executed, the wake bit is cleared and the HIBERNATE UUO returns immediately. The wake bit eliminates the problem of a job going to sleep and missing any wake conditions.

On a normal return, the job has been awakened.

3.2 CORE CONTROL

For various reasons, privileged jobs may desire to be locked in core so that they are never to be considered for swapping or shuffling. Some examples of these jobs are as follows:

Real-time jobs	These jobs require immediate access to the processor in response to an interrupt from an I/O device.
Display jobs	The display must be refreshed from a display buffer in the user's core area in order to keep the display picture flicker-free.
Batch	Batch throughput may be enhanced by locking the Batch job controller in core.
Performance analysis	Jobs monitoring the activities of the system need to be locked in core so that when they are entered to gather data, they are aware of their state and, therefore, can record activities of the monitor independent of the monitor.

3.2.1 DEFINITIONS

In swapping and non-swapping systems, unlocked jobs can occupy only the physical core not occupied by locked jobs. Therefore, locked jobs and timesharing jobs contend with one another for physical core memory. In order to control this contention, the system manager is provided with a number of system parameters as described below.

Total User Core is the physical core that can be used for locked and unlocked jobs. This value is equal to total physical core minus the monitor size.

CORMIN is the guaranteed amount of contiguous core that a single unlocked job can have. This value is a constant system parameter and is defined by the system manager at monitor generation time using MONGEN. It can be changed at monitor startup time using the ONCE ONLY dialogue. This value can range from 0 to Total User Core.

CORMAX is the largest contiguous size that an unlocked job can be. It is a time-varying system parameter that is reduced from its initial setting as jobs are locked in core. In order to satisfy the guaranteed size of CORMIN, the monitor never allows a job to be locked in core if this action would result in CORMAX becoming less than CORMIN. The initial setting of CORMAX is defined at monitor generation time using MONGEN and can be changed at monitor startup time using the ONCE ONLY dialogue. CORMAX can range from CORMIN to Total User Core. A guaranteed amount of core available for locked jobs can be made by setting the initial value of CORMAX to less than Total User Core.

3.2.2 LOCK AC, OR CALLI AC, 60

This UUO provides a mechanism for locking jobs in user memory. The user may specify if the high segment, low segment, or both segments are to be locked. When this UUO is executed by a privileged user program (privileges are granted by the system manager), it results in the job being locked in the optimal position in memory (at an extremity of user core).

A job may be locked in core if all of the following are true:

a. The job is privileged (privileges set from the accounting file ACCT.SYS by LOGIN).

b. The job, when locked, would not prevent another job from expanding to the guaranteed limit, CORMIN.

c. The job, when locked, would not prevent an existing job from running. Note that unlocked jobs can exceed CORMIN.

The call is:

```
MOVSI AC, 1          ;if high segment is to be locked
MOVSI AC, 0          ;if no high segment or if high
                     ;segment is not to be locked

HRRI  AC,1           ;if low segment is to be locked
HRRI  AC, 0          ;if low segment is not to be locked
LOCK AC,             ;or CALLI AC, 60
error return         ;AC contains an error code
normal return
```

On a normal return, the job is locked in core. If there is a high segment, the LH of AC contains its absolute address, shifted right nine bits. If there is no high segment, the LH of AC contains zero. The RH of AC contains the absolute address of the low segment, shifted right nine bits.

On an error return, the job is not locked in core and AC contains an error code indicating the condition that prevented the job from being locked. The error codes are as follows:

Error Code	Explanation
0	The UUO is not included in this system. The recommended procedure for determining if the LOCK UUO is implemented is to execute a GETTAB AC,,30 (CORTAB). An error return indicates that the LOCK UUO is not implemented.
1	The job does not have locking privileges.
2	If the job were locked in core, it would not be possible to run the largest existing non-locked job. (Applies only to swapping systems.)
3	If the job were locked in core, it would not be possible to meet the guaranteed largest size for an unlocked job, that is, CORMAX would be less than CORMIN.

NOTE

The CORE UUO may be given for the high or low segment of a locked job if the segment is not locked in core. When the segment is locked in core, the CORE UUO and the CORE command with a non-zero argument cannot be satisfied and, therefore, always give an erroneous response. The program should determine the amount of core needed for the execution and request this amount before executing the LOCK UUO.

Although memory fragmentation is minimized by both the LOCK UUO and the shuffler, the locking algorithm always allows job locking, even though severe fragmentation may take place, as long as

1) all existing jobs can continue to run, and
2) at least CORMIN is available as a contiguous space (see Figure 3-1E).

Therefore, it is important that system managers use caution when granting locking privileges. The following are guidelines for minimizing fragmentation when using the LOCK UUO.

3.2.2.1 Non-Swapping Systems - The guidelines for non-swapping systems are:

a. Any number of jobs can be locked in core without fragmentation occurring if the jobs are initiated immediately after the monitor is loaded.

b. During normal timesharing, a job is locked at the top of user core if the hole at the top of core is large enough to contain the job. Otherwise, the job is locked as low in core as possible.

c. Locking a job in core never makes the system fail, but it is possible that all of available core will not be utilized in some mixes of jobs.

3.2.2.2 Swapping Systems - The guidelines for swapping systems are:

a. There is no memory fragmentation if two jobs or less are locked in core.

b. There is no fragmentation if the locked jobs do not relinquish their locked status (i.e., no job terminates that has issued a LOCK UUO). In general, jobs with locking privileges should be production jobs.

c. If a job issuing a LOCK UUO is to be debugged and production jobs with locking privileges are to be run, the job to be debugged should be initiated and locked in core first, since it will be locked at the top of core. Then, the production jobs should be initiated since they will all be locked at the bottom of core. This procedure reserves the space at the top of core for the job being debugged and guarantees that there is no fragmentation as it locks and unlocks.

d. With a suitable setting of CORMIN and the initial setting of CORMAX in relation to Total User Core, the system manager can establish a policy which guarantees

   1) a maximum size for any unlocked job (CORMIN),

   2) a minimum amount of total lockable core for all jobs (Total User Core - CORMAX, and

   3) the amount of core which locked and unlocked jobs can contend for on a first-come-first-serve basis (Total User Core - initial CORMAX + CORMIN).

3.2.2.3 Core Allocation Resource - Because routines that lock jobs in core use the swapping and core allocation routines, they are considered a sharable resource. This resource is the semipermanent core allocation resource (mnemonic=CA). When a job issues a LOCK UUO and the system is currently engaged in executing a LOCK UUO for another job, the job enters the queue associated with the core allocation resource. Because a job may share a queue with other jobs and because swapping and shuffling may be required to position the job to where it is to be locked, the actual execution time needed to complete the process of locking a job might be on the order of seconds.

When it has been established that a job can be locked, the low segment number and the high segment number (if any) are stored as flags to activate the locking routines when the swapper and shuffler are idle. The ideal position for the locked job is also stored as a goal for the locking routines. In swapping systems, the ideal position is always achieved to guarantee minimum fragmentation. In non-swapping systems, minimum fragmentation is achieved only if the ideal position does not contain an active segment (see Figure 3-1).

In swapping systems, after the job is locked in core, the locking routine determines the size of the new largest contiguous region available to unlocked jobs. This value will be greater than or equal to CORMIN. If this region is less than the old value of CORMAX, then CORMAX is set equal to the size of the new reduced region. Otherwise, CORMAX remains set to its old value.

3.2.2.4 Unlocking Jobs - A job relinquishes its locked status when either the user program executes an EXIT or RESET UUO, or the monitor performs an implicit RESET for the user. Implicit RESETs occur when

a. The user program issues a RUN UUO, or

b. The user types any of the following monitor commands: R, RUN, GET, SAVE, SSAVE, CORE 0, and any system program-invoking command.

When the job is unlocked, it becomes a candidate for swapping and shuffling. CORMAX is increased to reflect the new size of the largest contiguous region available to unlocked jobs. However, CORMAX is never set to a greater value than its initial setting.

3.2.3 CORE AC, or CALLI AC, 11

This UUO provides a user program with the ability to expand and contract its core size as its memory requirements change. To allocate core in either or both segments, the left half of AC is used to specify the highest user address to be assigned to the high segment. If the left half of AC contains 0, the high segment core assignment is not changed. If the left half of AC is non-zero and is either less than 400000 or the length of the low segment, whichever is greater, the high segment is eliminated. If this is executed from the high segment, an illegal memory error message is printed when the monitor attempts to return control to the illegal address.

The error return is given if LH is greater than or equal to 400000 and if either the system does not have a two-segment capability or the user has been meddling without write access privileges (refer to Paragraph 6.2.3). An RH of 0 leaves the low segment core assignment unaffected. The monitor clears new core before assigning it to the user; therefore, privacy of information is ensured.

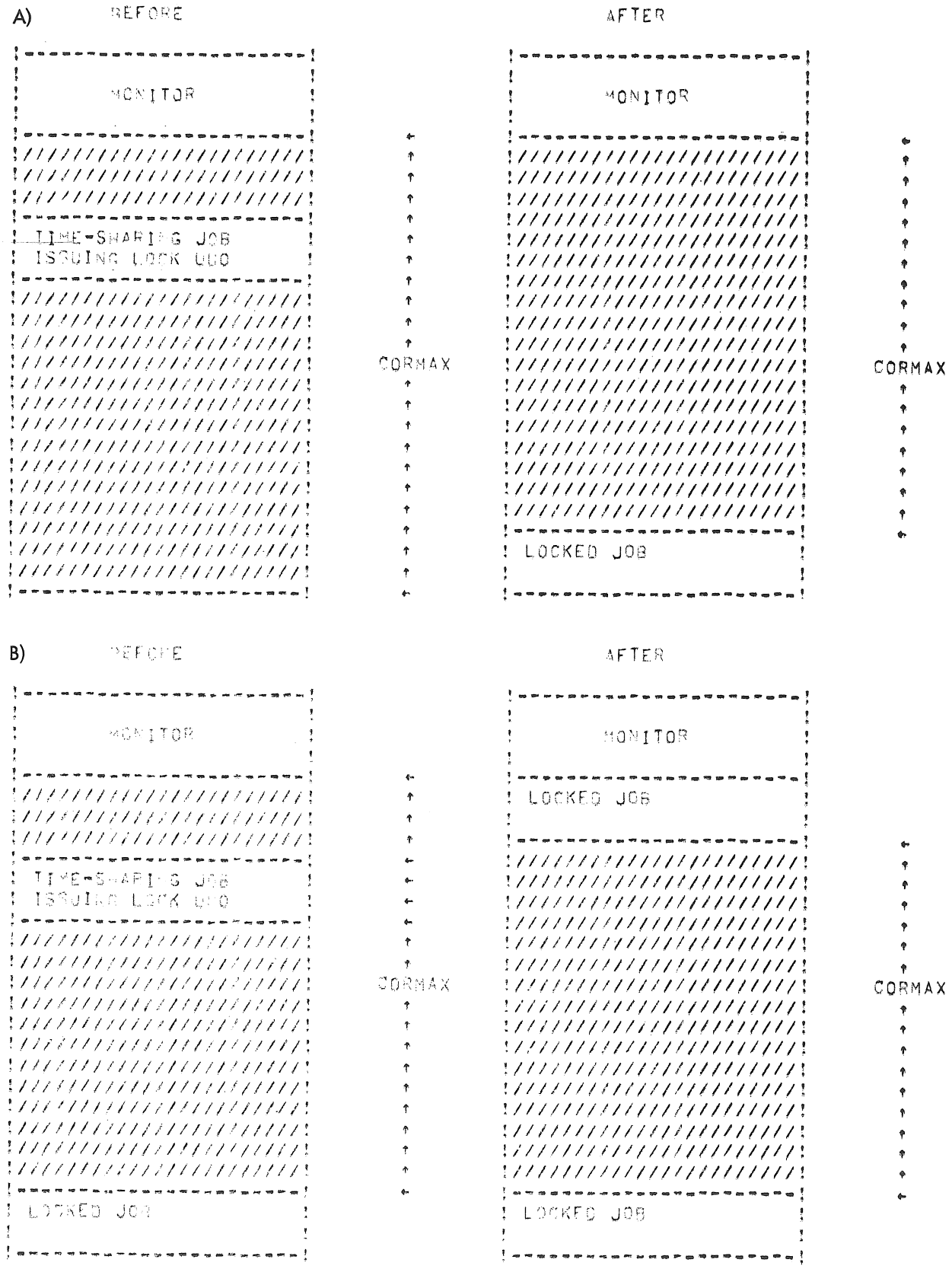

Figure 3-1 Locking Jobs In Core

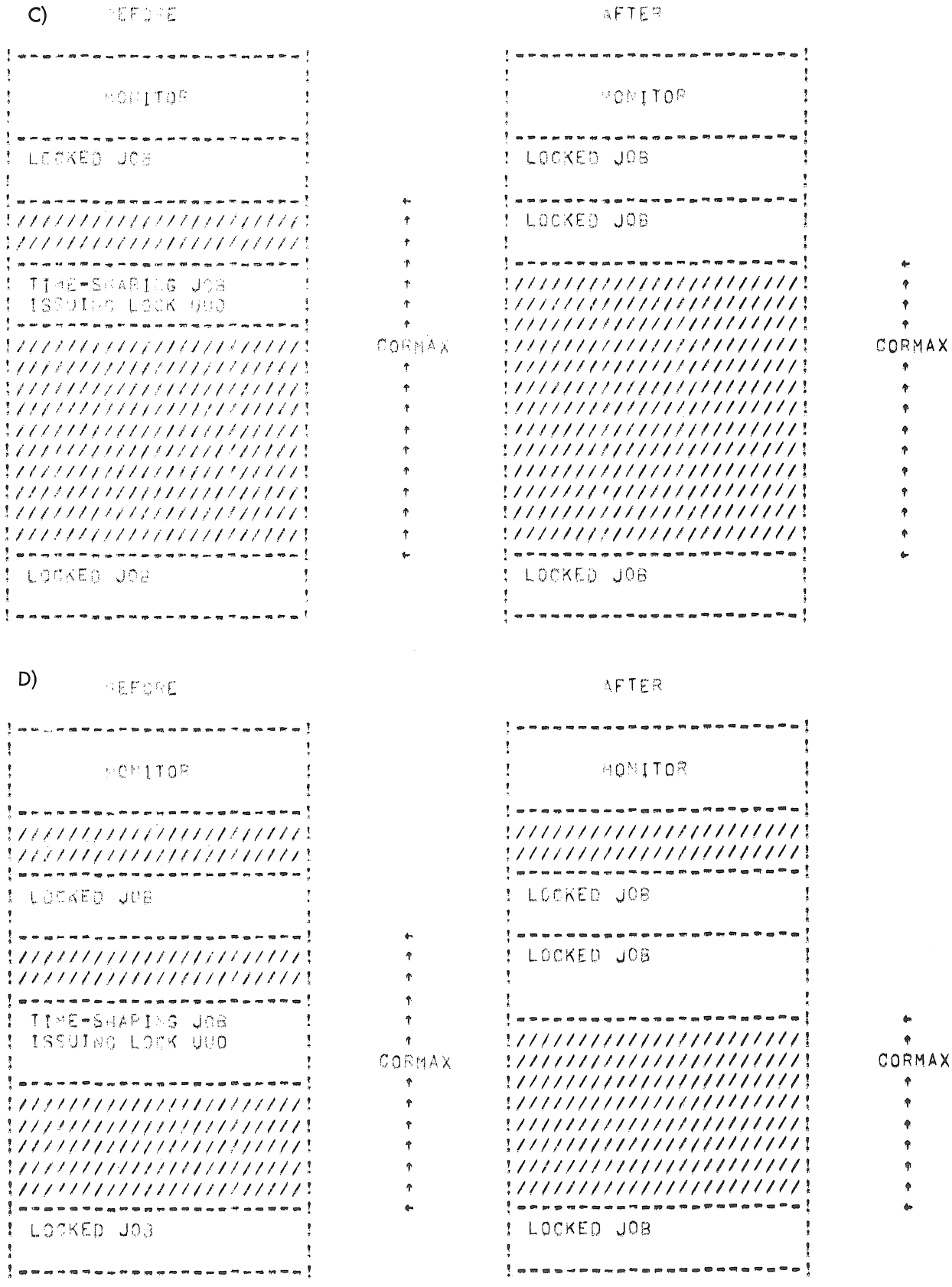

Figure 3-1 Locking Jobs In Core (Cont)

E) Unlikely Fragmentation Case

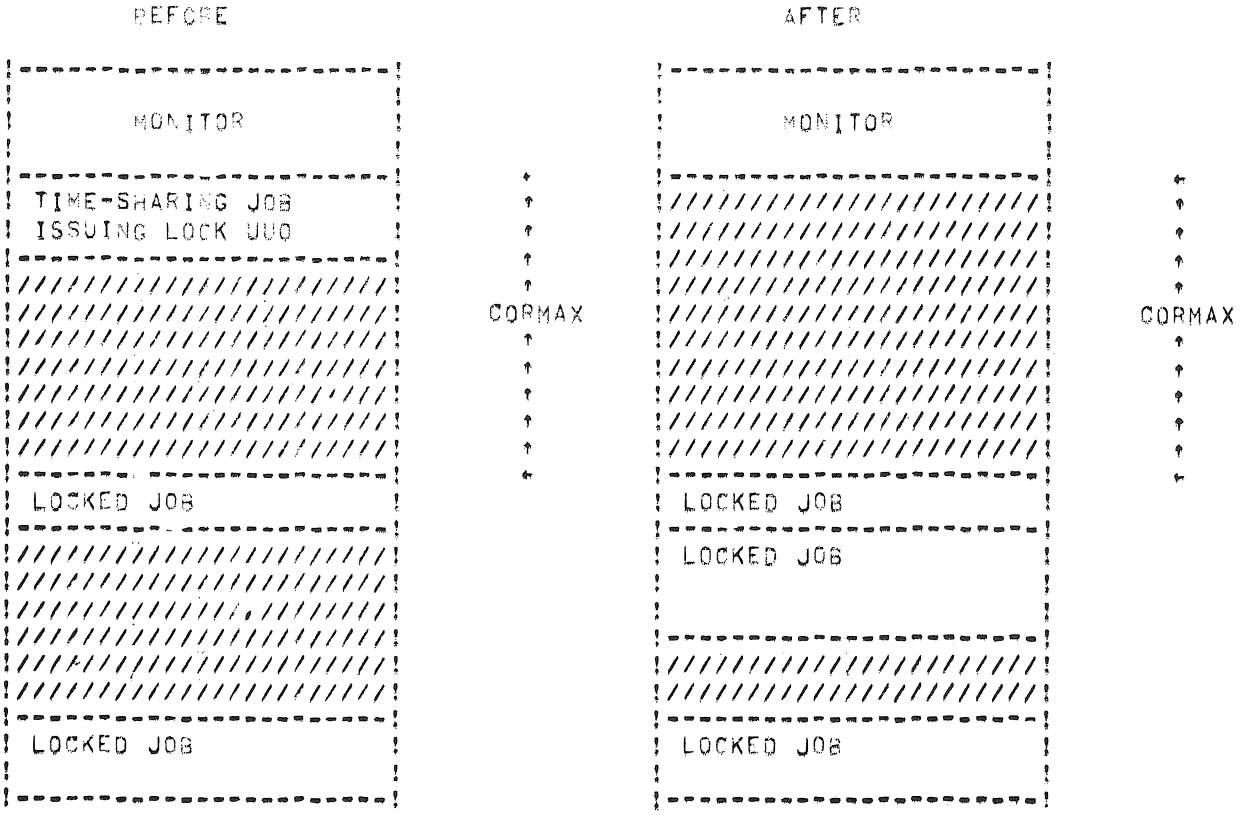

Figure 3-1 Locking Jobs In Core (Cont)

In swapping systems, this programmed operator returns the maximum number of 1K core blocks (all of core minus the monitor, unless an installation chooses to restrict the amount of core) available to the user. By restricting the amount of core available to users, the number of jobs in core simultaneously is increased. In nonswapping systems, the number of free and dormant 1K blocks is returned; therefore, the CORE UUO and the CORE command return the same information.

The call is:

```
MOVE AC [XWD HIGH ADR or 0, LOW ADDR or 0]
CORE AC,                                        ;or CALLI AC, 11
error return
normal return
```

The CORE UUO re-assigns the low segment (if RH is non-zero) and then re-assigns the high segment (if LH is non-zero). If the sum of the new low segment and the old high segment exceeds the maximum amount of core allowed to a user, the error return is given, the core assignment is unchanged, and the maximum core available to the user for high and low segments (in 1K blocks) is returned in the AC. In a nonswapping system, the number of free and dormant 1K blocks is returned.

If the sum of the new low segment and the new high segment exceeds the maximum amount of core allowed to a user, the error return is given, the new low segment is assigned, the old high segment remains, and the maximum core available to the user in 1K blocks is returned in the AC. Therefore, to increase the low segment and decrease the high segment at the same time, two separate CORE UUOs should be used to reduce the chances of exceeding the maximum size allowed to a user job.

If the new low segment extends beyond 377777, the high segment shifts up into the virtual addressing space instead of being overlaid. If a long low segment is shortened to 377777 or less, the high segment shifts from the virtual addressing space to 400000 instead of growing longer or remaining where it was. If the high segment is a program, it does not execute properly after a shift unless it is a self-relocating program in which all transfer instructions are indexed.

If the high segment is eliminated by a CORE UUO, a subsequent CORE UUO, in which the LH is greater than 400000, will create a new, nonsharable segment rather than re-establishing the old high segment. This segment becomes sharable after it has been:

a. Given an extension .SHR.
b. Written onto the storage device.
c. Closed so that a directory entry is made.
d. Initialized from the storage device by GET, R, or RUN commands or RUN or GETSEG UUOs.

The loader and the SAVE and GET commands use the above sequence to create and initialize new sharable segments.

3.2.4 SETUWP AC, or CALLI AC, 36

This UUO allows a user program to set or clear the hardware user-mode write protect bit and to obtain the previous setting. It must be used if a user program is to modify the high segment.

The call is:

```
SETUWP AC,                    ;OR CALLI AC, 36
error return
normal return
```

If the system has a two-register capability, the normal return will be given unless the user has been meddling without write privileges, in which case an error return will be given. A normal return is given whether or not the program has a high segment, because the reentrant software is designed to allow users to write programs for two-register machines, which will run under one-register machines. Compatibility of source and relocatable binary files is, therefore, maintained between one-register and two-register machines.

If the system has a one-register capability, the error return (bit 35 of AC=0) is given. This error return allows the user program to find out whether or not the system has a two-segment capability. The user program specifies the setting of the user-mode write protect bit in bit 35 of AC (write protect = 1, write privilieges = 0). The previous setting of the user-mode write protect bit is returned in bit 35 of AC, so that any user subroutine can preserve the previous setting before changing it. Therefore, nested user subroutines, which either set or clear the bit, can be written, provided the subroutines save the previous value of the bit and restore it on returning to its caller.

3.6 ENVIRONMENTAL INFORMATION

3.6.1 Timing Information

The central processor clock, which generates interrupts at the power-source frequency (60 Hz in North America, 50 Hz in most other countries), keeps time in the monitor. Each clock interrupt (tick) corresponds to 1/60th (or 1/50th) of a second of elapsed real time. The clock is set initially to the current time by terminal input when the system is started, as is the current date. When the clock reaches midnight, it is reset to zero, and the date is advanced.

3.6.1.1 DATE AC, or CALLI AC, 14 - A 12-bit binary integer computed by the formula

$$date=((year-1964)\times 12+(month-1))\times 31+day-1$$

represents the date.

This integer representation is returned right justified in AC.

3.6.1.2 TIMER AC, or CALLI AC, 22 - This UUO returns the time of day, in clock ticks (jiffies), right justified in AC. The MSTIME UUO should normally be used so that the time is not a function of the cycle.

3.6.1.3 MSTIME AC, or CALLI AC, 23 - This UUO returns the time of day, in milliseconds, right justified in AC.

3.6.2 Job Status Information

3.6.2.1 RUNTIM AC, or CALLI AC, 27 - The accumulated running time (in milliseconds) of the job number specified in AC is returned right justified in AC. If the job number in AC is zero, the running time of the currently running job is returned. If the job number in AC does not exist, zero is returned.

3.8 REAL-TIME PROGRAMMING

3.8.1 RTTRP AC, or CALLI AC, 57

The real-time trapping UUO is used by timesharing users to dynamically connect real-time devices to the priority interrupt system, to respond to these devices at interrupt level, to remove the devices from the interrupt system, and to change the PI level on which the devices are associated. The RTTRP UUO can be called from UUO level or from interrupt level. This is a privileged UUO that requires the job to have real-time privileges (granted by LOGIN) and to be locked in core (accomplished by LOCK UUO). These real-time privileges are assigned by the system manager and obtained by the monitor from ACCT.SYS. The privilege bits required are:

1) PVLOCK (Bit 14) - allows the job to be locked in core.
2) PVRTT (Bit 13) - allows the RTTRP UUO to be executed.

WARNING

Improper use of features of the RTTRP UUO can cause the TOPS-10 system to fail in a number of ways. Since design goals of this UUO were to give the user as much flexibility as possible, some system integrity had to be sacrificed. The most common errors are protected against since user programs run in user mode with all ACs saved. It is recommended that debugging of real-time programs not be done when system integrity is important. However, once these programs are debugged, they can run simultaneously with batch and timesharing programs.

Real-time jobs control devices one of two ways: block mode or single mode. In block mode, an entire block of data is read before user's interrupt program is run. In single mode, the user's interrupt program is run every time the device interrupts. Furthermore, there are two types of block mode: fast block mode and normal block mode. These differ in response times. The response time to read a block of data in fast block mode is 6.5 μs per word and in normal block mode, 14.6 μs per word. (This is the CPU time to complete each data transfer.) In all modes, the response time measured from the receipt of the real-time device interrupt to the start of the user control program is 100 μs.

The RTTRP UUO allows a real-time job to either put a BLKI or BLKO instruction directly on a PI level (block mode) or add a device to the front of the monitor PI channel CONSO skip chain (single mode). When an interrupt occurs from the real-time device in single mode or at the end of a block of data in block mode, the monitor saves the current state of the machine (the ACs, APR flags, protection-relocation register, UUO trap addresses 40 and 41, and the reserved instruction trap addresses 60 and 61), sets the new protection-relocation register and APR flags, and traps to the user's interrupt routine. The user services his device and then returns control to the monitor to restore the previous state of the machine and to dismiss the interrupt.

In fast block mode the monitor places the BLKI/BLKO instruction directly in the PI trap location followed by a JSR to the context switcher. This action requires that the PI channel be dedicated to the real-time job during any transfers. In normal block mode the monitor places the BLKI/BLKO instruction directly after the real-time device's CONSO instruction in the CONSO skip chain (refer to Chapter 7).

Any number of real-time devices using either single mode or normal block mode can be placed on any available PI channel. The average extra overhead for each real-time device on the same channel is 5.5 μs per interrupt.

The call is:

```
        MOVEI AC, RTBLK         ;AC contains address of data block.
        RTTRP AC,               ;or CALLI AC, 57, put device on PI level.
        error return            ;AC contains an error code.
        normal return           ;PI is set up properly.
```

The data block depends on the mode used. In single mode the data block is:

```
RTBLK:  XWD PICHL, TRPADR       ;PI channel (1-6) and trap address.
        EXP APRTRP              ;APR trap address.
        CONSO DEV, BITS         ;CONSO chain instruction.
        0                       ;no BLKI/BLKO instruction.
```

The data block in fast block mode is:

```
RTBLK:  XWD PICHL, TRPADR       ;PI and trap address when BLKO done.
        EXP APRTRP              ;APR trap address.
        BLKO DEV, BLKADR        ;BLKI or BLKO instruction.
        0                       ;BLKADR points to the IOWD of
                                ;block to be sent.
```

The data block in normal block mode is:

```
RTBLK:  XWD PICHL, TRPADR       ;channel and trap address.
        EXP APRTRP              ;APR trap address.
        CONSO DEV, @BITMSK      ;control bit mask from user area.
        BLKI DEV, BLKADR        ;BLKI instruction.
```

3.8.1.1 Data Block Mnemonics - The following mnemonics are used in describing the data block associated with the RTTRP UUO.

PICHL - PICHL is the PI level on which the device is to be placed. Levels 1-6 are legal depending on the system configuration. If PICHL = 0, the device is removed from all levels. When a device is placed on a PI level, normally all other occurrences of the device on any PI level are removed. If the user desires the same device on more than one PI level simultaneously (i.e., a data level and an error level), he can issue the RTTRP UUO with PICHL negative. This indicates to the system that any other occurrence of this device (on any PI level) is not to be removed. Note that this addition to a PI level counts as a real-time device, occupying one of the possible real-time device slots.

TRPADR - TRPADR is the location trapped to by the real-time interrupt (JRST TRPADR). Before the trap occurs, all ACs are saved by the monitor and can be overwritten without concern for their contents.

APRTRP - APRTRP is the trap location for all APR traps. When an APR trap occurs, the monitor simulates a JSR APRTRP. The user gains control from an APR trap on the same PI level that his real-time device is on. The monitor always traps to the user program on illegal memory references, non-existent memory

references, and push-down overflows. This allows the user to properly turn off his real-time device if needed. The monitor also traps on the conditions specified by the APRENB UUO (see Paragraph 3.1.3.1). No APR errors are detected if the interrupt routine is on a PI level higher than or equal to the APR interrupt level.

DEV - DEV is a real-time device code.

BITS - BITS is the bit mask of all interrupt bits of the real-time device and must not contain any other bits. If the user desires control of this bit mask from his user area, he may specify one level of indirection in the CONSO instruction (no indexing), i.e., CONSO DEV, @ MASK where MASK is the location in the user area of the bit mask. MASK must not have any bits set in the indirect or index fields.

BLKADR - BLKADR is the address in the user's area of the BLKI/BLKO pointer word. Before returning to the user, the monitor adds the proper relocation factor to the right half of the pointer word. Data can only be read into the low segment above the protected job data area, i.e., above location 114. Since the pointer word is in the user's area, the user can set up a new pointer word when the word count goes to 0 at interrupt level. This allows fast switching between buffers. When the user desires to set up his own pointer word, the right half of the word must be set as an absolute address instead of a relative address. The job's relocation value is returned from both the LOCK UUO and the first RTTRP UUO executed for setting the BLKI/BLKO instruction. If this pointer word does not contain a legal address, a portion of the system might be overwritten. A check should be made to determine if the negative word count in the left half of the pointer word is too large. If the word count extends beyond the user's own area, the device may cause a non-existent memory interrupt, or may overwrite a timesharing job. If all of the above precautions are taken, this method of setting up the pointer word is much faster and more flexible than issuing the RTTRP UUO at interrupt level.

3.8.1.2 Interrupt Level Use of RTTRP - The format of the RTTRP UUO at interrupt level is similar to the format at user level except for two restrictions:

1) AC 16 and AC 17 cannot be used in the UUO call (i.e., CALLI 16,57 is illegal at interrupt level).
2) All ACs are overwritten when the UUO is executed at interrupt level. Therefore, the user must save any desired ACs before issuing the RTTRP UUO. This restriction is used to save time at interrupt level.

CAUTION

If an interrupt level routine executes a RTTRP UUO that affects the device currently being serviced, no additional UUOs of any kind (including RTTRP) can be executed during the remainder of the interrupt. At this point, any subsequent UUO dismisses the interrupt.

3.8.1.3 RTTRP Returns - On a normal return, the job is given user IOT privileges. These privileges allow the user to execute all restricted instructions including the necessary I/O instructions to control his device.

The IOT privilege must be used with caution because improper use of the I/O instructions could halt the system (i.e., CONO APR,0, CONO PI,0, or HALT). Note that a user can obtain just the user IOT privilege by issuing the RTTRP UUO with PICHL = 0.

An error return is not given to the user until RTTRP scans the entire data block to find as many errors as possible. On return, AC may contain the following error codes.

Code	Value	Meaning
Bit 26=1	1000	Device already in use by another job.
Bit 27=1	400	Illegal AC used during RTTRP UUO at interrupt level.
Bit 28=1	200	Job not locked in core, or not privileged.
Bit 29=1	100	System limit for real-time devices exceeded.
Bit 30=1	40	Illegal format of CONSO, BLKO, or BLKI instruction.
Bit 31=1	20	BLKADR or pointer word illegal.
Bit 32=1	10	Error address out of bounds.
Bit 33=1	4	Trap address out of bounds.
Bit 34=1	2	PI channel not currently available for BLKI/BLKO's.
Bit 35=1	1	PI channel not available (restricted use by system).

3.8.1.4 Restrictions -

1) Devices may be chained onto any PI channel that is not used for BLKI/BLKO instructions by the system or by other real-time users using fast block mode. This includes the APR channel. Normally PI levels 1 and 2 are reserved by the system for magnetic tapes and DECtapes. PI level 7 is always reserved for the system.

2) Each device must be chained onto a PI level before the user program issues the CONO DEV, PIA to set the device onto the interrupt level. Failure to observe this rule or failure to set the device on the same PI level that was specified in the RTTRP UUO could hang the system.

3) If the CONSO bit mask is set up and one of the corresponding flags in a device is on, but the device has not been physically put on its proper PI level, a trap may occur to the user's interrupt service routine. This occurs because there is a CONSO skip chain for each PI level, and if another device interrupts whose CONSO instruction is further down the chain than that of the real-time device, the CONSO associated with the real-time device is executed. If one of the hardware device flags is set and the corresponding bit in the CONSO bit mask is set, the CONSO skips and a trap occurs to the user program even through the real-time device was not causing the interrupt on that channel. To avoid this situation the user can keep the CONSO bit mask in his user area (refer to Paragraph 3.8.1.1). This procedure allows the

(continued on next page)

user to chain a device onto the interrupt level, keeping the CONSO bit mask zero until the device is actually put on the proper PI level with a CONO instruction. This situation never arises if the device flags are turned off until the CONO DEV, PIA can be executed.

4) The user should guard against putting programs on high priority interrupt levels which execute for long periods of time. These programs could cause real-time programs at lower levels to lose data.

5) The user program must not change any locations in the protected job data area (locations 20-114), because the user is running at interrupt level and full context switching is not performed.

6) If the user is using the BLKI/BLKO feature, he must restore the BLKI/BLKO pointer word before dismissing any end-of-block interrupts. This is accomplished with another RTTRP UUO or by directly modifying the absolute pointer word supplied by the first RTTRP UUO. Failure to reset the pointer word could cause the device to overwrite all of memory.

3.8.1.5 Removing Devices from a PI Channel - When PICHL=0 in the data block (see Paragraph 3.8.1.1), the device specified in the CONSO instruction is removed from the interrupt system. If the user removes a device from a PI chain, he must also remove the device from the PI level (CONO DEV, 0).

A RESET, EXIT, or RUN UUO from the timesharing levels removes all devices from the interrupt levels (see Paragraph 3.2.2.4). These UUOs cause a CONO DEV, 0 to be executed before the device is removed. Monitor commands that issue implicit RESETS also remove real-time devices (e.g., R, RUN, GET, CORE 0, SAVE, SSAVE).

3.8.1.6 Dismissing the Interrupt - The user program must always dismiss the interrupt in order to allow monitor to properly restore the state of the machine. The interrupt may be dismissed with any UUO other than the RTTRP UUO or any instruction that traps to absolute location 60. The standard method of dismissing the interrupt is with a UJEN instruction (op code 100). This instruction gives the fastest possible dismissal by trapping to location 60.

3.8.1.7 Examples

```
********** EXAMPLE 1 **********
          SINGLE MODE

TITLE RTSNGL - PAPER TAPE READ TEST USING CONSO CHAIN

PIOFF=400                ;TURN PI SYSTEM OFF
PION=200                 ;TURN PI SYSTEM ON
TAPE=400                 ;NO MORE TAPE IN READER IF TAPE=0
BUSY=20                  ;DEVICE IS BUSY READING
DONE=10                  ;A CHARACTER HAS BEEN READ
```

(continued on next page)

```
PDATA:  Z                       ;LOCATION WHERE DATA IS READ INTO

PTRTST: RESET                   ;RESET THE PROGRAM
        MOVE [XWD 1,1]          ;LOCK BOTH HIGH AND LOW SEGMENTS
        LOCK                    ;LOCK THE JOB IN CORE
        JRST FAILED             ;LOCK UUO FAILED
        SETZM PTRCSO            ;MAKE SURE CONSO BITS ARE ZERO
        SETZM DONFLG            ;INITIALIZE DONE FLAG
        MOVEI RTBLK             ;GET ADDRESS OF REAL TIME DATA BLOCK
        RTTRP                   ;PUT REAL TIME DEVICE ON THE PI LEVEL
        JRST FAILED             ;RTTRP UUO FAILED
        MOVEI 1,DONE            ;SET UP CONSO BIT MASK
        HLRZ 2,RTBLK            ;GET PI NUMBER FROM RTBLK
        TRO 2,BUSY              ;SET UP CONO BITS TO START TAPE GOING
        CONO PI,PIOFF           ;GUARD AGAINST ANY INTERRUPTS
        MOVEM 1,PTRCSO          ;STORE CONSO BIT MASK
        CONO PTR,(2)            ;TURN PTR ON
        CONO PI,PION            ;ALLOW INTERRUPTS AGAIN
        MOVEI 5                 ;SET UP TO SLEEP FOR 5 SECONDS
        CALLI 31                ;SLEEP
        SKIPN DONFLG            ;HAVE WE FINISHED READING THE TAPE
        JRST .-3                ;NO GO BACK TO SLEEP
        EXIT

RTBLK:  XWD 5,TRPADR            ;PI CHANNEL AND TRAP ADDRESS
        EXP APRTRP              ;APR ERROR TRAP ADDRESS
        CONSO PTR,@PTRCSO       ;INDIRECT CONSO BIT MASK = PTRCSO
        Z                       ;NO BLKI/O INSTRUCTION

PTRCSO: Z                       ;CONSO BIT MASK
DONFLG: Z                       ;PI LEVEL TO USER LEVEL COMM.
RTBLK1: Z                       ;DATA BLOCK TO REMOVE PTR
        Z                       ;FROM PI CHANNEL
        CONSO PTR,0
        Z

TRPADR: CONSO PTR,TAPE          ;END OF TAPE?
        JRST TDONE              ;YES, GO STOP JOB
        DATAI PTR,PDATA         ;READ IN DATA WORD
        UJEN                    ;DISMISS THE INTERRUPT

APRTRP: Z                       ;APR ERROR TRAP ADDRESS
TDONE:  MOVEI RTBLK1            ;SET UP TO REMOVE PTR
        CONO PTR,0              ;TAKE DEVICE OFF HARDWARE PI LEVEL
        RTTRP                   ;REMOVE FROM SOFTWARE PI LEVEL
        JFCL                    ;IGNORE ERRORS
        SETOM DONFLG            ;MARK THAT READ IS OVER
        SETZM PTRCSO            ;CLEAR CONSO BIT MASK
        UJEN                    ;DISMISS THE INTERRUPT

FAILED: TTCALL 3,[ASCIZ/RTTRP UUO FAILED!/]
        EXIT

        END PTRTST
```

********** EXAMPLE 2 **********
FAST BLOCK MODE

```
        TITLE RTFBLK - PAPER TAPE READ TEST IN BLKI MODE

        TAPE=400                ;NO MORE TAPE IN READER IF TAPE=0
        BUSY=20                 ;DEVICE IS BUSY READING
        DONE=10                 ;A CHARACTER HAS BEEN READ
```

(continued on next page)

```
BLKTST: RESET                   ;RESET THE PROGRAM
        MOVE [XWD 1,1]          ;LOCK BOTH HIGH AND LOW SEGMENTS
        LOCK                    ;LOCK THE JOB IN CORE
        JRST FAILED             ;LOCK UUO FAILED
        SETZM DONFLG            ;INITIALIZE DONE FLAG
        MOVEI RTBLK             ;GET ADDRESS OF REAL TIME DATA BLOCK
        RTTRP                   ;PUT REAL TIME DEVICE ON THE PI LEVEL
        JRST FAILED             ;RTTTP UUO FAILED
        HLRZ 2,RTBLK            ;GET PI NUMBER FROM RTBLK
        TRO 2,BUSY              ;SET UP CONO BITS TO START TAPE GOING
        CONO PTR,(2)            ;TURN PTR ON
        MOVEI 5                 ;SET UP TO SLEEP FOR 5 SECONDS
        CALLI 31                ;SLEEP
        SKIPN DONFLG            ;HAVE WE FINISHED READING THE TAPE
        JRST .-3                ;NO GO BACK TO SLEEP
        EXIT

RTBLK:  XWD 6,TRPADR            ;PI CHANNEL AND TRAP ADDRESS
        EXP APRTRP              ;APR ERROR TRAP ADDRESS
        BLKI PTR,POINTR         ;READ A BLOCK AT A TIME
        Z

POINTR: IOWD 5,TABLE            ;POINTER FOR BLKI INSTRUCTION
OPOINT: IOWD 5,TABLE            ;ORIGINAL POINTER WORD FOR BLKI
TABLE:  BLOCK 5                 ;TABLE AREA FOR DATA BEING READ
DONFLG: Z                       ;PI LEVEL TO USER LEVEL COMM.
RTBLK1: Z                       ;DATA BLOCK TO REMOVE PTR
        Z                       ;FROM PI CHANNEL
        CONSO PTR,Ø
        Z

TRPADR: CONSO PTR,TAPE          ;END OF TAPE?
        JRST TDONE              ;YES, GO STOP JOB
        MOVE OPOINT             ;GET ORIGINAL POINTER WORD
        MOVEM POINTR            ;RESTORE BLKI POINTER WORD
        UJEN                    ;DISMISS THE INTERRUPT

APRTRP: Z                       ;APR ERROR TRAP ADDRESS
TDONE:  MOVEI RTBLK1            ;SET UP TO REMOVE PTR
        CONO PTR,Ø              ;TAKE DEVICE OFF HARDWARE PI LEVEL
        RTTRP                   ;REMOVE FROM SOFTWARE PI LEVEL
        JFCL                    ;IGNORE ERRORS
        SETOM DONFLG            ;MARK THAT READ IS OVER
        UJEN                    ;DISMISS THE INTERRUPT

FAILED: TTCALL 3,[ASCIZ/RTTRP UUO FAILED!/]
        EXIT

        END BLKTST


********** EXAMPLE 3 **********
        NORMAL BLOCK MODE

        TITLE RTNBLK - PAPER TAPE READ TEST IN BLKI MODE

        TAPE=400                ;NO MORE TAPE IN READER IF TAPE=Ø
        BUSY=20                 ;DEVICE IS BUSY READING
        DONE=10                 ;A CHARACTER HAS BEEN READ
```

(continued on next page)

```
BLKTST: RESET                   ;IO RESET
        MOVE [XWD 1,1]          ;LOCK BOTH HIGH AND LOW SEGMENTS
        LOCK                    ;LOCK THE JOB IN CORE
        JRST FAILED             ;LOCK UUO FAILED
        MOVEI RTBLK1            ;GET ADDRESS OF REAL TIME BLOCK
        RTTRP                   ;GET USER IOT PRIVILEGE
        JRST FAILED             ;UUO FAILED!
        CONO PTR,0              ;CLEAR ALL PTR FLAGS
        SETZM DONFLG            ;INITIALIZE DONE FLAG
        MOVEI RTBLK             ;GET ADDRESS OF REAL TIME DATA BLOCK
        RTTRP                   ;PUT REAL TIME DEVICE ON THE PI LEVEL
        JRST FAILED             ;RTTRP UUO FAILED
        MOVE POINTR             ;GET RELOCATED POINTER WORD FOR LATER
        MOVEM OPOINT            ;STORE FOR INTERRUPT LEVEL USE
        HLRZ 2,RTBLK            ;GET PI NUMBER FROM RTBLK
        TRO 2,BUSY              ;SET UP CONO BITS TO START TAPE GOING
        CONO PTR,(2)            ;TURN PTR ON
        MOVEI 5                 ;SET UP TO SLEEP FOR 5 SECONDS
        SLEEP
        SKIPN DONFLG            ;HAVE WE FINISHED READING THE TAPE
        JRST .-3                ;NO GO BACK TO SLEEP
        EXIT

RTBLK:  XWD 6,TRPADR            ;PI CHANNEL AND TRAP ADDRESS
        EXP APRTRP              ;APR ERROR TRAP ADDRESS
        CONSO PTR,DONE          ;WAIT ONLY FOR DONE FLAG
        BLKI PTR,POINTR         ;READ A BLOCK AT A TIME

POINTR: IOWD 5,TABLE            ;POINTER FOR BLKI INSTRUCTION
OPOINT: Z
TABLE:  BLOCK 5                 ;TABLE AREA FOR DATA BEING READ
DONFLG: Z                       ;PI LEVEL TO USER LEVEL COMM.
RTBLK1: Z                       ;DATA BLOCK TO REMOVE PTR
        Z                       ;FROM PI CHANNEL
        CONSO PTR,0
        Z

TRPADR: CONSO PTR,TAPE          ;END OF TAPE?
        JRST TDONE              ;YES, GO STOP JOB
        MOVE OPOINT             ;GET ORIGINAL POINTER LOCATION
        MOVEM POINTR            ;STORE IN POINTER LOCATION
        UJEN                    ;DISMISS THE INTERRUPT

APRTRP: Z                       ;APR ERROR TRAP ADDRESS
TDONE:  MOVEI RTBLK1            ;SET UP TO REMOVE PTR
        CONO PTR,0              ;TAKE DEVICE OFF HARDWARE PI LEVEL
        RTTRP                   ;REMOVE FROM SOFTWARE PI LEVEL
        JFCL                    ;IGNORE ERRORS
        SETOM DONFLG            ;MARK THAT READ IS OVER
        UJEN                    ;DISMISS THE INTERRUPT

FAILED: TTCALL 3,[ASCIZ/RTTRP UUO FAILED!/]
        EXIT

        END BKJTST
```

3.8.2 RTTRP Executive Mode Trapping

In special cases, the real-time user requires a faster response time than that offered by the RTTRP UUO when executed in user mode. To accommodate these cases, the user can specify a special status bit in the RTTRP UUO call, which gives the program control in exec mode (refer to Paragraph 2.1.3). Exec-mode trapping gives response times of less than 10 μs to real-time interrupts. To use this exec-mode trapping, the job must have real-time privileges (granted by LOGIN) and be locked in core (accomplished by the LOCK UUO). The privilege bits required are:

1) PVTRPS (Bit 15)
2) PVLOCK (Bit 14)
3) PVRTT (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 ACs and with the processor in exec mode. Therefore, the user program must save and restore all ACs that are used, must not execute any UUOs, and cannot leave exec mode. This means that the programs must be self-relocating (i.e., through the use of an index or base register).

CAUTION

Improper use of the exec mode feature of the RTTRP UUO can cause the TOPS-10 system to fail in a number of ways. Unlike the user mode feature of RTTRP, errors are not protected against since the programs run in exec mode with no ACs saved.

To specify RTTRP exec-mode trapping, bit 17 of the second word in the data block (RTBLK) must be set to 1. This implies that no context switching is to be done and that a JRST TRPADR is to be used to enter the user's real-time interrupt routine. The user program must save and restore all ACs and should dismiss the interrupt with a JRSTF @ TRPADR. 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 UUO returns the absolute addresses 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 UUO: single mode, normal block mode, and fast block mode.

3.8.2.1 Example

```
        TITLE RTEXEC

        PIA=5
        DONE=10
        BUSY=20
        TAPE=400
        I=1
        AC=2
        OPDEF HIBERNATE [CALLI 72]

RTEXEC: RESET                   ;RESET THE PROGRAM
        SETZM DONFLG            ;INITIALIZE THE DONE FLAG
        MOVE AC,[XWD 1,1]
        LOCK AC,                ;LOCK THE JOB IN CORE
                                ;ABSOLUTE ADDRESS OF JOB IS RETURNED
                                ;IN AC
        JRST FAILED             ;ERROR RETURN
        HRRZS AC                ;GET ONLY LOW SEGMENT ADDRESS
        LSH AC,9                ;JUSTIFY ADDRESS
        MOVEM AC,INDEX          ;SAVE BASE ADDRESS FOR USE AT INTERRUPT
                                ;LEVEL
        ADDM AC,EXCHWD          ;RELOCATE INTERRUPT LEVEL PROGRAM
        ADDM AC,JENWD           ;RELOCATE RETURN INSTRUCTION
        MOVEI AC,RTBLK          ;CONNECT REAL TIME DEVICE
        RTTRP AC,               ;TO THE PI SYSTEM
        JRST FAILED             ;RTTRP UUO FAILED
        CONO PTR,20+PIA         ;START REAL TIME DEVICE READING
SLEEP:  MOVEI AC,^D1000         ;SLEEP
        HIBERNATE AC,           ;FOR 10 MILLISECONDS
        JRST FAILED             ;FAILED
        SKIPN DONFLG            ;IS THE INTERRUPT LEVEL PROGRAM DONE
        JRST SLEEP              ;NO, GO BACK TO SLEEP
        EXIT                    ;YES, EXIT

RTBLK:  XWD PIA,TRPADR
        XWD 1,APRTRP            ;BIT 17 SAYS TRAP IN EXEC MODE
        CONSO PTR,DONE
        0

TRPADR: 0                       ;JSR TRPADR IS DONE UPON 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 IN THE NEXT CHARACTER
RETURN: EXCH I,INDEX(I)         ;RESTORE AC'S USED
JENWD:  JRSTF @TRPADR           ;DISMISS THE INTERRUPT

APRTRP: 0                       ;APR ERRORS WILL TRAP HERE
TDONE:  CONO PTR,0              ;TAKE THE READER OFF LINE
        SETOM DONFLG(I)         ;MARK THAT THE TAPE IS FINISHED
        JRST RETURN(I)          ;GO DISMISS THE INTERRUPT

FAILED: TTCALL 3,[ASCIZ/UUO FAILURE/]
        EXIT

DONFLG: 0                       ;FLAG TO SPECIFY END OF JOB
PDATA:  0                       ;DATA WORD
INDEX:  0                       ;BASE INDEX REGISTER

        END RTEXEC
```

3.8.3 TRPSET AC, or CALLI AC, 25

The TRPSET feature may be used to guarantee some of the fast response requirements of real-time users. In order to achieve fast response to interrupts, this feature temporarily suspends the running of other jobs during its use. This limits the class of problems to be solved to cases where the user wants to transfer data in short bursts at predefined times. Therefore, because the data transfers are short, the time during which timesharing is stopped is also short, and the pause probably will not be noticed by the timesharing users.

The TRPSET UUO allows the user program to gain control of the interrupt locations. If the user does not have the TRPSET privileges (PVTRPS, bit 15), an error return to the next location after the CALLI is always given, and the user remains in user mode. Timesharing is turned back on. If the user has the TRPSET privileges, the central processor is placed in user I/O mode. If AC contains zero, timesharing is turned on if it was turned off. If the LH of AC is within the range 40 through 57, all other jobs are stopped from being scheduled and the specified executive PI location (40-57) is patched to trap directly to the user. In this case, the monitor moves the contents of the relative location specified in the right half of AC, adds the job relocation address to the address field, and stores it in the specified executive PI location.

Thus, the user can set up a priority interrupt trap into his relocated core area. On a normal return, AC contains the previous contents of the address specified by LH of AC, so that the user program may restore the original contents of the PI location when the user is through using this UUO. If the LH of AC is not within the range 40 through 57, an error return is given just as if the user did not have the privileges. The call is:

```
        MOVE AC,[XWD N, ADR]
        TRPSET AC,
        ERROR RETURN
        NORMAL RETURN
        .
        .
        .
ADR:    JSR TRAP                ;Instruction to be stored
                                ;in exec PI location
                                ;after relocation added to it.
TRAP:   Ø                       ;Here on interrupt from exec.
```

The monitor assumes that user ADR contains either a JSR U or BLKI U, where U is a user address; consequently, the monitor adds the job relocation to the contents of location U to make it an absolute IOWD. Therefore, a user should reset the contents of U before every TRPSET call.

A RESET UUO returns the user to normal user mode. The following instruction sequence is used to place the real-time device RTD on channel 3.

```
INT 46:   BLKI RTD,INBLOK          ;relocation constant
                                   ;for user is added
INT 47:   JSR XITINT               ;to RH when instructions
          .                        ;are placed into 46 and 47.
          .
          .
START:    MOVEI AC,INT46
          HRLI AC,46
          TRPSET AC,
          JRST EXITR               ;error return
          AOBJN AC, .+1            ;normal return
          TRPSET AC,
          JRST EXITR               ;error return
          .                        ;normal return
          .
          .
          .
XITINT:   0                        ;PC saved
          . . .                    ;interrupt dismiss routine
```

If the interrupt occurs while some other part of the user's program is running, the user may dismiss from the interrupt routine with a JEN @ XITINT. However, if the machine is in exec mode, a JEN instruction issued in user mode does not work because of memory relocation. This is solved by a call to UJEN (op code 100). This UUO causes the monitor to dismiss the interrupt from exec mode. In this case, the address field of the UJEN instruction is the user location when the return PC is stored (i.e., UJEN XITINT). The following sequence enables the user program to decide whether it can issue a JEN to save time or dismiss the interrupt with a UUO call.

```
XITINT:   0                        ;PC with bits in LH

          JRST 1,.+1               ;essential instruction.
                                   ;returns machine to
                                   ;user mode.
          MOVEM AC, SAVEAC         ;save accumulator AC
          .                        ;service interrupt here
          .
          .

          MOVE AC, XITINT          ;get PC with bits
          SETZM EFLAG

          TLNN AC, 10000           ;was machine in user
                                   ;mode at entry?
          SETOM EFLAG              ;no
          MOVE AC, SAVEAC          ;RESTORE saved AC
          SKIPE EFLAG
          UJEN XITINT              ;not in user mode at entry
          JEN @ XITINT

SAVEAC:   0

EFLAG:    0
```

On entering the routine from absolute 47 with a JSR to XITINT + REL (where REL. is the relocation constant), the processor enters exec mode. The first executed instruction in the user's routine must, therefore, reset the user mode flag, thereby enabling relocation and protection. The user must proceed with caution when changing channel interrupt chains under timesharing, making certain that the real-time job can co-exist with other timesharing jobs.

3.8.4 UJEN (Op Code 100)

This op code dismisses a user I/O mode interrupt if one is in progress. If the interrupt is from user mode, a JRST 12, instruction dismisses the interrupt. If the interrupt came from executive mode, however, this operator is used to dismiss the interrupt. The monitor restores all accumulators, and executes JEN @ U where user location U contains the program counter as stored by a JSR instruction when the interrupt occurred.

3.8.5 HPQ AC, OR CALLI AC, 71*

The HPQ UUO is used by privileged users to place their jobs in a high-priority scheduler run queue. These queues are always scanned by the scheduler before the normal run queues, and any runnable job in one of these queues is executed before all other jobs in the system. Thus, real-time associated jobs can receive fast response from the timesharing scheduler.

Jobs in high-priority queues are not examined for swap-out until all other queues have been scanned. If a job in a high-priority queue must be swapped, the lowest priority job is transferred first, and the highest priority job last. If the highest priority job is swapped, then that job is the first to be swapped in for immediate execution. Therefore, in addition to being scanned before all other queues for job execution, the high-priority queues are examined after all other queues for swap-out and before all queues for swap-in.

The HPQ UUO requires as an argument the high-priority queue number of the queue to be entered. The lowest high-priority queue is 1, and the highest priority queue is equivalent to the number of queues that the system is built for. The call is as follows:

```
MOVE AC, HPQNUM          ;get high-priority queue number
HPQ AC,                  ;or CALLI AC, 71
error return
normal return
```

* Also available as a console command.

On an error return, AC contains -1 if the user did not have the correct privileges. The privilege bits are 6 through 9 in the privilege word (JBTPRV). These four bits specify a number from 0-17 octal, which is the highest priority queue attainable by the user.

On a normal return, the job is in the desired high-priority queue. A RESET or an EXIT UUO or a queue number of 0 as an argument places the job back to the timesharing level.

7.1 JOB SCHEDULING

The number of jobs that may be run simultaneously must be specified in creating the TOPS-10 Monitor. Up to 127 jobs may be specified. Each user accessing the system is assigned a job number.

In a multiprogramming system all jobs reside in core, and the scheduler decides what jobs should run. In a swapping system, jobs exist on an external storage device (usually disk or drum) as well as in core. The scheduler decides not only what job is to run but also when a job is to be swapped out onto the disk (drum) or brought back into core.

In a swapping system, jobs are retained in queues of varying priorities that reflect the status of the jobs at any given moment. Each job number possible in the system resides in only one queue at any time. A job may be in one of the following queues:

a. Run queues - for runnable jobs waiting to execute. (There are three run queues of different levels of priorities.)

b. I/O wait queue - for jobs waiting while doing I/O.

c. I/O wait satisfied queue - for jobs waiting to run after finishing I/O.

d. Sharable device wait queue - for jobs waiting to use sharable devices.

e. TTY wait queue - for jobs waiting for input or output on the user's console.

f. TTY wait satisfied queue - for jobs that completed a TTY operation and are awaiting action.

g. Stop queue - for processes that have been completed or aborted by an error and are awaiting a new command for further action.

h. Null queue - for all job numbers that are inactive (unassigned).

Each queue is addressed through a table. The position of a queue address in a table represents the priority of the queue with respect to the other queues. Within each queue, the position of a job determines its priority with respect to the other jobs in the same queue. The status of a job is changed when it is placed in a different queue.

Each job, when it is assigned to run, is given a quantum time. When the quantum time expires, the job ceases to run and moves to a lower priority run queue. The activities of the job currently running may cause it to move out of the run queue and enter one of the wait queues. For example: when a currently running job begins input from a DECtape, it is placed in the I/O wait queue, and the input is begun. A second job is set to run while the input of the first job proceeds. If the second job then decides to access a DECtape for an I/O operation, it is stopped because the DECtape control is busy, and it is put in the queue for jobs waiting to access the DECtape control. A third job is set to run. The input operation of the first job finishes, making the DECtape control available to the second job. The I/O operation of the second job is initiated, and the job is transferred from the device wait queue to the I/O wait queue. The first job is transferred from the I/O wait queue to the highest priority run queue. This permits the first job to preempt the running of the third job. When the quantum time of the first job becomes zero, it is moved into the second run queue, and the third job runs again until the second job completes its I/O operations.

Data transfers also use the scheduler to permit the user to overlap computation with data transmission. In unbuffered modes, the user supplies an address of a command list containing pointers to relative locations in the user area to and from which data is to be transferred. When the transfer is initiated, the job is scheduled into an I/O wait queue where it remains until the device signals the scheduler that the entire transfer has been completed.

In buffered modes, each buffer contains a use bit to prevent the user and the device from using the same buffer at the same time (refer to Paragraph 4.3). If the user overtakes the device and requires the buffer currently being used by the device as his next buffer, the user's job is scheduled into an I/O wait queue. When the device finishes using the buffer, the device calls the scheduler to reactivate the job. If the device overtakes the user, the device is stopped at the end of the buffer and is restarted when the user finishes with the buffer.

Scheduling occurs at each clock tick (1/60th or 1/50th of a second) or may be forced at monitor level between clock ticks if the current job becomes blocked (unrunnable). The asynchronous swapping algorithm is also called at each clock tick and has the task of bringing a job from disk into core. This function depends on

a. The core shuffling routine, which consolidates unused areas in core to make sufficient room for the incoming job,
b. The swapper, which creates additional room in core by transferring jobs from core to disk.

Therefore, when the scheduler is selecting the next job to be run, the swapper is bringing the next job to be run into core. The transfer from disk to core takes place while the central processor continues computation for the previous job.

7.2 PROGRAM SWAPPING

Program swapping is performed by the monitor on one or more units of the system independent of the file structures that may also use the units. Swapping space is allocated and deallocated in clusters of 1K words (exactly); this size is the increment size of the memory relocation and protection mechanism. Directories are not maintained, and retrieval information is retained in core. Most user segments are written onto the swapping units as contiguous units. Swapping time and retrieval information is, therefore, minimized. Segments are always read completely from the swapping unit into core with one I/O operation. The swapping space on all units appears as a single system file, SWAP.SYS, in directory SYS in each file structure. This file is protected from all but privileged programs by the standard file protection mechanism (refer to Paragraph 6.2.3).

The reentrant capability reduces the demands on core memory, swapping space, swapping channel, and storage channel; however, to reduce the use of the storage channel, copies of sharable segments are kept on the swapping device. This increases the demand for swapping space. To prevent the swapping space from being filled by user's files and to keep swapped segments from being fragmented, swapping space is preallocated when the file structure is refreshed. The monitor dynamically achieves the space-time balance by assuming that there is no shortage of swapping space. Swapping space is never used for anything except swapped segments, and the monitor keeps a single copy of as many segments as possible in this space. (The maximum number of segments that may be kept may be increased by individual installations but is always at least as great as the number of jobs plus one.) If a sharable segment on the swapping space is currently unused, it is called a dormant segment. An idle segment is a sharable segment that is not used by users in core; however, at least one swapped-out user must be using the segment or it would be a dormant segment.

Swapping disregards the grouping of similar units into file structures; therefore, swapping is done on a unit basis rather than a file structure basis. The units for swapping are grouped in a sorted order, referred to as the active swapping list. The total virtual core, which the system can allocate to users, is equal to the total swapping space preallocated on all units in the active swapping list. In computing virtual core, sharable segments count only once, and dormant segments do not count at all. The monitor does not allow more virtual core to be granted than the system has capacity to handle.

When the system is started, the monitor reads the home blocks on all the units that it was generated to handle. The monitor determines from the home blocks which units are members of the active swapping list. This list may be changed at once-only time. The change does not require refreshing of the file structures, as long as swapping space was preallocated on the units when they were refreshed. All of the units with swapping space allocated need not appear in the active swapping list. For example: a drum and disk pack system should have swapping space allocated on both drum and disk packs. Then, if the drum becomes inoperable, the disk packs may be used for swapping without refreshing.

Users cannot proceed when virtual core is exhausted; therefore, FILSER is designed to handle a variety of disks as swapping media. The system administrator allocates additional swapping space on slower disks and virtually eliminates the possibility of exhausting virtual core; therefore, in periods of heavy demand, swapping is slower for segments that must be swapped on the slower devices. It is also undesirable to allow dormant segments to take up space on high-speed units. This forces either fragmentation on fast units or swapping on slow units; therefore, the allocation of swapping space is important to overall system efficiency.

The swapping allocator is responsible for assigning space for the segment the swapper wants to swap out. It must decide

a. Onto which unit to swap the segment.
b. Whether to fragment the unit if not enough contiguous space is available.
c. Whether to make room by deleting a dormant segment.
d. Whether to use a slower unit.

The units in the active swapping list are divided into swapping classes, usually according to device speed. For simplicity, the monitor assumes that all the units of class 0 are first followed by all the units of class 1. Swapping classes are defined when the file structures are refreshed and may be changed at once-only time.

When attempting to allocate space to swap out a low or high segment, the monitor performs the following:

Step	Procedure
1	The monitor looks for contiguous space on one of the units of the first swapping class.
2	The monitor looks for noncontiguous space on one of the units in the same class.
3	The monitor checks whether deleting one or more dormant segments would yield enough contiguous or noncontiguous space.

If all of these measures fail, the monitor repeats the process on the next swapping class in the active swapping list. If none of the classes yield enough space, the swapper begins again and deletes enough dormant segments to fragment the segment across units and classes. When a deleted segment is needed again, it is retrived from the storage device.

7.3 DEVICE OPTIMIZATION

7.3.1 Concepts

Each I/O operation on a unit consists of two steps: positioning and data transferring. To perform I/O, the unit must be positioned, unless it is already on a cylinder or is a non-positioning device. To position a unit, the controller cannot be performing a data transfer. If the controller is engaged in a data transfer, the positioning operation of moving the arm to the desired cylinder cannot begin until the data transfer is complete.

The controller ensures that the arms have actually moved to the correct cylinder. This check is called verification, and the time required is fixed by hardware. If verification fails, the controller interrupts the processor, and the software recalibrates the positioner by moving it to a fixed place and beginning again. When verification is complete, the controller reads the sector headers to find the proper sector on which to perform the operation. This operation is called searching. Finally, the data is transferred to or from the desired sectors. To understand the optimization, the transfer operation includes verification, searching, and the actual transfer. The time from the initiation of the transfer operation to the actual beginning of the transfer is called the latency time. The channel is busy with the controller for the entire transfer time; therefore, it is important for the software to minimize the latency time.

The FILSER code, a routine that queues disk requests and makes optimization decisions, handles any number of channels and controllers and up to eight units for each controller. Optimization is designed to keep:

a. As many channels as possible performing data transfers at the same time.

b. As many units positioning on all controllers, which are not already in position for a data transfer.

Several constraints are imposed by the hardware. A channel can handle only one data transfer on one control at a time. Furthermore, the control can handle a data transfer on only one of its units at a time. However, the other units on the control can be positioning while a data transfer is taking place provided the positioning commands were issued prior to the data transfer. Positioning requests for a unit on a controller that is busy doing a data transfer for another of its units must be queued until the data transfer is finished. When a positioning command is given to a unit through a controller, the controller is busy for only a few microseconds; therefore, the software can issue a number of positioning commands to different units as soon as a data transfer is complete. All units have only positioning mechanism that reaches each point; therefore, only one positioning operation can be performed on a unit at the same time. All other positioning requests for a unit must be queued.

The software keeps a state code in memory for each active file, unit, controller, and channel, to remember the status of the hardware. Reliability is increased because the software does not depend on the status information of the hardware. The state of a unit is as follows:

I	Idle; No positions or transfers waiting or being performed.
SW	Seek Wait; Unit is waiting for control to become idle so that it can initiate positioning (refer to Paragraph 6.2).
S	Seek; Unit is positioning in response to a SEEK UUO; no transfer of data follows.
PW	Position Wait; Unit is waiting for control to become idle so that it can initiate positioning.
P	Position; Unit is positioning; transfer of data follows although not necessarily on this controller.
TW	Transfer Wait; Unit is in position and is waiting for the controller/channel to become idle so that it can transfer data.
T	Transfer; Unit is transferring; the controller and channel are busy performing the operation.

Table 7-1 lists the possible states for files, units, controllers, and channels.

Table 7-1
Software States

File†	Unit	Controller	Channel
I	I	I	I
	SW		
	S		
PW	PW		
P	P		
TW	TW		
T	T	T	T

†Cannot be in S or SW state because SEEKs are ignored if the unit is not idle.

7.3.2 Queuing Strategy

When an I/O request for a unit is made by a user program because of an INPUT or OUTPUT UUO, one of several things can happen at UUO level before control is returned to the buffer-strategy module in UUOCON, which may, in turn, pass control back to the user without rescheduling. If an I/O request requires positioning of the unit, either the request is added to the end of the position-wait queue for

the unit if the control or unit is busy, or the positioning is initiated immediately. If the request does not require positioning, the data is transferred immediately. If the channel is busy, the request is added to the end of the transfer-wait queue for the channel. The control gives the processor an interrupt after each phase is completed. Optimization occurs at interrupt level when a position-done or transfer-done interrupt occurs.

7.3.2.1 Position-Done Interrupt Optimization - The following action occurs only if a transfer-done interrupt does not occur first. Data transfer is started on the unit unless the channel is busy transferring data for some other unit or control. If the channel is busy, the request goes to the end of the transfer-wait queue for that channel.

7.3.2.2 Transfer-Done Interrupt Optimization - When a transfer-done interrupt occurs, all the position-done interrupts inhibited during the data transfer are processed for the controller, and the requests are placed at the end of the transfer-wait queue for the channel. All units on the controller are then scanned. The requests in the position-wait queues on each unit are scanned to see the request nearest the current cyclinder. Positioning is begun on the unit of the selected request. All requests in the transfer-wait queue for all units on the channel that caused the interrupt are then scanned and the latency time is measured. The request with the shortest latency time is selected, and the new transfer begins.

7.3.3 Fairness Considerations

When the system selects the best task to run, users making requests to distant parts of the disk may not be serviced for a long time. The disk software is designed to make a fair decision for a fixed percentage of time. Every n decisions the disk software selects the request at the front of the position-wait or transfer-wait queue and processes it, because that request has been waiting the longest. The value of n is set to 10 (decimal) and may be changed by redefining symbols with MONGEN (see MONITR.OPR).

7.3.4 Channel Command Chaining

7.3.4.1 Buffered Mode - Disk accesses are reduced by using the chaining feature of the data channel. Prior to reading a block in buffered mode, the device independent routine checks to see if there is another empty buffer, and if the next relative block within the file is a consecutive logical block within the unit. If both checks are true, FILSER creates a command list to read two or more consecutive blocks into scattered core buffers. Corresponding decisions are made when writing data in buffered mode, and, if possible, two or more separate buffers are written in one operation. The command chaining decision is not made when a request is put into a position-wait or transfer-wait queue;

instead, it is postponed until the operation is performed, thus increasing the chances that the user program will have more buffers available for input or output.

7.3.4.2 Unbuffered Mode - Unbuffered modes do not use channel chaining, and therefore, read or write one command word at a time. Each command word begins at the beginning of a 128-word block. If a command word does not contain an even multiple of 128 words, the remaining words of the last block are not read, if reading, and are written with zeroes, if writing.

7.4 MONITOR ERROR HANDLING

The monitor detects a number of errors. If a hardware error is detected, the monitor repeats the operation ten times. If the failure occurs eleven times in a row, it is classified as a hard error. If the operation succeeds after failing one to ten times, it is a soft error.

7.4.1 Hardware Detected Errors

Hardware detected errors are classified either as device errors or as data errors. A device error indicates a malfunction of the controller or channel. A data error indicates that the hardware parity did not check or a search for a sector header either did not succeed or had bad parity (the user's data is probably bad).

A device error sets the IODERR bit in the channel status word, and a data error sets the IODTER bit.

Disk units may have imperfect surfaces; therefore, a special non-timesharing diagnostic program, MAP, is provided to initially find all the bad blocks on a specified unit. The logical disk addresses of any bad regions of one or more bad blocks are recorded in the bad allocation table (BAT) block on the unit. The timesharing monitor allocates all storage for files; therefore, it uses the BAT block to avoid allocating blocks that have previously proven bad. The MAP program writes two copies of the BAT block because the BAT block might be destroyed. If the MAP program is not used, the monitor discovers the bad regions when it tries to use them and adds this information to the BAT block. However, the first user of the bad region loses that part of his data.

A hard data error usually indicates a bad surface; therefore, the monitor never returns the bad region to free storage. This results in the bad region causing an error only once. The bad unit and the logical disk address are stored in the retrieval information block (RIB) of the file when the file is CLOSEd or RESET and the extent of the bad region is determined. The origin and length of the bad region is stored in the bad allocation table (BAT) block.

7.4.2 Software Detected Errors

The monitor makes a number of software checks on itself. It checks the folded checksum (refer to Appendix I) computed for the first word of every group and stored in the retrieval pointer. The monitor also checks for inconsistencies when comparing locations in the retrieval information block with expected values (filename, filename extension, project-programmer number, special code, logical block number). The monitor checks for inconsistencies in the storage allocation table block when comparing the number of free clusters expected with the number of zeroes. A checksum error or an inconsistency error in the SAT block or RIB normally indicates that the monitor is reading the wrong block. When these errors occur, the monitor sets the improper mode error bit (IOIMPM) in the user channel status word and returns control to the user program.

7.5 DIRECTORIES

7.5.1 Order of Filenames

The names of newly created files are appended to the directory if the directory does not contain more than 64 filenames. If the directory contains more than 64 filenames, a second block is used for the new filenames. When filenames are deleted from the first block, entries from the second block are not moved into the first. When additional new files are created, their names are added to the end of the first block of the directory instead of the end of the directory. Thus, the order of the filenames in the directory may not be according to the date of creation.

7.5.2 Directory Searches

Table space in core memory is used to reduce directory searching times. The JBTPPB table contains pointers to a list of four-word blocks for the user's project-programmer number, one block for each file structure on which the user has a UFD.

Four-word name and access blocks contain copies of LOOKUP information for recently-accessed files and may reduce disk accesses to one directory read for a LOOKUP on a recently-active file. Recent LOOKUP failures are also kept in core, but are deleted when space is needed.

7.6 PRIORITY INTERRUPT ROUTINES

7.6.1 Channel Interrupt Routines

Each of the seven PI channels has two abolute locations associated with it in memory: 40+2n and 41+2n, where n is a channel number (1-7). When an interrupt occurs on a channel, control is immediately transferred to the first of the two associated locations (unless an interrupt on a higher priority

channel is being processed). For fast service of a single device, the first location contains either a BLKI or BLKO instruction. For service of more than one device on the same channel, the first location contains a JSR to location CHn in the appropriate channel interrupt routine. The JSR ensures that the current state of the program counter is saved.

Each channel interrupt routine (mnemonic name, CHANn, where n is the channel number) consists of three separate routines:

CHn:	The contents of the program counter is saved in location CHn. CHn+1 contains a JRST to the first device service routine in the interrupt chain.
SAVCHn:	The routine to save the contents of a specified number of accumulators. It is called from the device service routines with a JSR.
XITCHn:	The routine to restore saved accumulators. Device service routines exit to XITCHn with a POPJ PDP, if SAVCHn was previously called.

7.6.2 Interrupt Chains

Each device routine contains a device interrupt routine DEVINT where DEV is the three-letter mnemonic for the device concerned. This routine checks to determine whether an interrupt was caused by device DEV. The interrupt chain of a given channel is a designation for the logical positioning of each device interrupt routine associated with that channel.

The monitor flow of control on the interrupt level through a chain is illustrated below. Channel 5 is used in the example.

Monitor Routine	Relevant Code	Explanation
Absolute Locations	52/JSR CH5 53/	;control transferred here ;on interrupt
	↓	
CHAN5	CH5: 0	;contents of PC saved here
	JRST PTPINT	;control transfers to first
	↓	;link in interrupt chain
	PTPINT: CONSO PTP,PTPDON	;if PDP done bit is
PTPSER	JRST LPTINT	;on, PTP was cause
	⋮	;of interrupt -
	⋮	;otherwise, go to
	↓	;next device.
	LPTINT: CONSO LPT,LPTLOV+LPTERR+LPTDON	
LPTSER	JEN @ CH5	;three possible bits
	⋮	;may indicate that
	⋮	;LPT caused interrupt

When a real-time device is added to the interrupt chain (CONSO skip chain) by a RTTRP UUO (refer to Paragraph 3.8.1), the device is added to the front of the chain. After putting a real-time device on Channel 5 in single mode (refer to Paragraph 3.8.1), the chain is as follows:

Monitor Routine	Relevant Code	Explanation
Absolute Locations	52/JSR CH5 53/	;control transferred here ;on interrupt
	↓	
CHAN5	CH5: 0	;contents of PC saved here
	JRST RDTINT	;control transfers to first
	↓	;link in interrupt chain
RTDEV	RTDINT: CONSO RTD,BITS	
	JRST PTPINT	
	JRST <context switcher and dispatch for real-time interrupts >	
	PTPINT: CONSO PTP,PTPDON	;if PTP done bit is
PTPSER	JRST LPTINT	;on, PTP was cause
	:	;of interrupt -
	:	;otherwise, go to
	↓	;next device.
	LPTINT:CONSO LPT, LPTLOV+LPTERR+LPTDON	
LPTSER	JEN @ CH5	;three possible bits
	:	;may indicate that
	:	;LPT caused interrupt

After putting a real-time device on channel 5 in normal block mode (refer to Paragraph 3.8.1), the chain is as follows:

Monitor Routine	Relevant Code	Explanation
Absolute Locations	52/JSR CH5 53/	;control transferred here ;on interrupt
	↓	
CHAN5	CH5: 0	;contents of PC saved here
	JRST RTDINT	;control transfers to first
	↓	;link in interrupt chain
RTDEV	RTDINT:CONSO RTD,BITS	
	JRST PTPINT	
	BLKI RTD,POINTR	
	JRST <context switcher >	
	JEN @ CH5	

(continued on next page)

Monitor Routine	Relevant Code	Explanation
	PTPINT: CONSO PTP,PTPDON	;if PTP done bit is
PTPSER	JRST LPTINT	;on, PTP was cause
	:	;of interrupt -
	:	;otherwise, go to
	↓	;next device.
	LPTINT:CONSO LPT, LPTLOV+LPTERR+LPTDON	
LPTSER	JEN @ CH5	;three possible bits
	:	;may indicate that
	:	;LPT caused interrupt.

After putting a real-time device on channel 6 in fast block mode (refer to Paragraph 3.8.1), the chain is as follows:

Monitor Routine	Relevant Code	Explanation
Absolute Locations	54/BLKO RTD,POINTR 55/JSR CH6	;control transferred ;here on interrupt
CHAN6	CH6: 0 JRST <context switcher>	;contents of PC saved ;control transfers to ;context switcher.

The exec mode trapping feature can be used with any of the standard forms of the RTTRP UUO: single mode, normal block mode, and fast block mode. The following examples illustrate the chain when used with each of the three modes.

Single Mode (Exec Mode)

Monitor Routine	Relevant Code	Explanation
Absolute Locations	52/JSR CH5 53/	;control transferred here ;on interrupt
	↓	
CHAN5	CH5: 0 JRST RDTINT	;contents of PC saved here ;control transfers to first
	↓	;link in interrupt chain
RTDEV	RTDINT: CONSO RTD,BITS JRST PTPINT JSR TRPADR JEN @ CH5	

(continued on next page)

Single Mode (Exec Mode) (Cont)

Monitor Routine	Relevant Code	Explanation
	PTPINT:CONSO PTP,PTPDON	;if PTP done bit is
PTPSER	JRST LPTINT	;on, PTP was cause
	:	;of interrupt -
	:	;otherwise, go to
	↓	;next device.
	LPTINT:CONSO LPT, LPTLOV+LPTERR+LPTDON	
LPTSER	JEN @ CH5	;three possible bits
	:	;may indicate that
	:	;LPT caused interrupt

Normal Block Mode (Exec Mode)

Monitor Routine	Relevant Code	Explanation
Absolute	52/JSR CH5	;control transferred here
Locations	53/	;on interrupt
	↓	
CHAN5	CH5: 0	;contents of PC saved here
	JRST RTDINT	;control transfers to first
	↓	;link in interrupt chain
RTDEV	RTDINT:CONSO RTD,BITS	
	JRST PTPINT	
	BLKI RTD,POINTR	
	JSR TRPADR	
	JEN @ CH5	
	PDPINT: CONSO PTP,PTPDON	;if PTP done bit is
PTPSER	JRST LPTINT	;on, PDP was cause
	:	;of interrupt -
	:	;otherwise, go to
	↓	;next device.
	LPTINT:CONSO LPT,LPTLOV+LPTERR+LPTDON	
LPTSER	JEN @ CH5	;three possible bits
	:	;may indicate that
	:	;LPT caused interrupt.

Fast Block Mode (Exec Mode)

Monitor Routine	Relevant Code	Explanation
Absolute	54/BLKO RTD,POINTR	;control transferred here
Locations	55/JSR CH6	;on interrupt
	↓	

(continued on next page)

Fast Block Mode (Exec Mode) (Cont)

Monitor Routine	Relevent Code	Explanation
CHAN6	CH6: 0 JRST RDTINT ↓	;contents of PC saved here ;control transfers to first ;link in interrupt chain
RTDEV	RTDINT: JSR TRPADR JEN @ CH6	

7.7 MEMORY PARITY ERROR RECOVERY

The memory parity error recovery code allows the machine to run with PARITY STOP up, thereby gaining 10% more CPU speed than with PARITY STOP enabled. This procedure differentiates between user mode and executive mode when a parity error occurs. If the machine was in user mode, the current job is stopped, and the word causing bad parity is rewritten with good parity. The following message is typed on the user's terminal:

```
?ERROR IN JOBn
?MEM PAR AT USER pppppp; BAD WORD ddddddddddd
    AT USER adr = ABS. xxxxxx
```

and the following message is typed simultaneously on device OPR, interrupting any current typeout:

```
?USER MODE PAR ERROR AT ABS LOC xxxxxx FOR JOBn
```

where

n is the job number of the current job
pppppp is the user PC when the parity error occurred
ddddddddddd is the bad data word read (expressed in octal)
adr is the user address of the bad word
xxxxxx is the absolute address of the bad word

If the machine was in executive mode when the parity error occurred, recovery is not attempted since a monitor routine has read bad data. The following message (with no carriage return, line feed following) is typed on CTY and the machine HALTS:

```
EXEC PARITY ERROR STOP
```

At this point the operator must depress the PARITY STOP key and hit CONTINUE. The machine should stop almost immediately with a memory failure. If the parity error is not reproducible on a memory scan, the following message is typed on the CTY on the same line as the previous message and the machine HALTS with the PC at 777777:

----SPURIOUS

System reload is required after an executive mode memory parity failure.

The algorithm for determining the bad memory location in both executive and user mode is to scan core from location 20_8 through location C (MEMSIZ). In both modes, the parity error is detected at APR interrupt level. For executive mode the memory scan when CONTINUE is hit runs at APR level. For user mode a clock level interrupt is requested, and the memory scan and subsequent typeouts are processed at this level. The following two counters are kept for user mode parity analysis:

PARTOT - the total number of user mode parity errors since system was loaded

PARSPR - the number of errors for which recovery failed (no parity error on memory scan) and the job was not stopped.

Also the counters PARPC, PARADR, and PARWRD contain the user PC, the absolute location, and the bad data word, respectively, of the most recent user mode memory parity error.

digital

DIGITAL EQUIPMENT CORPORATION, Maynard, Massachusetts, Telephone: (617) 897-5111 • ARIZONA, Phoenix • CALIFORNIA, Anaheim, Los Angeles, Oakland, Palo Alto • COLORADO, Denver • CONNECTICUT, Meriden • DISTRICT OF COLUMBIA, Washington (College Park, Md.) • FLORIDA, Orlando • GEORGIA, Atlanta • ILLINOIS, Chicago • INDIANA, Indianapolis • MASSACHUSETTS, Cambridge and Waltham • MICHIGAN, Ann Arbor • MINNESOTA, Minneapolis • MISSOURI, St. Louis • NEW JERSEY, Parsippany and Princeton • NEW MEXICO, Albuquerque • NEW YORK, Centereach (L.I.), New York City, (Englewood, N.J.), and Rochester • NORTH CAROLINA, Durham/Chapel Hill • OHIO, Cleveland and Dayton • OREGON, Portland • PENNSYLVANIA, Philadelphia and Pittsburgh • TENNESSEE, Knoxville • TEXAS, Dallas and Houston • UTAH, Salt Lake City • WASHINGTON, Seattle • ARGENTINA, Buenos Aires • AUSTRALIA, Brisbane, Melbourne, Perth and Sydney • BELGIUM, Brussels • CANADA, Edmonton, Alberta; Vancouver, British Columbia; Carleton Place, Ottawa and Toronto, Ontario; and Montreal, Quebec • CHILE, Santiago • ENGLAND, Birmingham, London, Manchester and Reading • FRANCE, Paris • GERMANY, Cologne, Hanover, Frankfurt and Munich • ITALY, Milan • JAPAN, Tokyo • NETHERLANDS, The Hague • SWEDEN, Stockholm • SWITZERLAND, Geneva and Zurich • PHILIPPINES, Manila • VENEZUELA, Caracas