XVM/RSX PART VI
INPUT/OUTPUT OPERATIONS

CHAPTER 1

RSX INPUT/OUTPUT

1.1 INTRODUCTION TO RSX I/O

Under RSX, I/O is flexible and essentially device-independent. Although certain operations require the identification of a specific device, most user and system requests are for logical units that are temporarily associated with particular physical devices. Requests for service are queued and subsequently processed according to the priority of the requesting task. Because the hardware I/O Processor runs in parallel with the Central Processor Unit, the system is capable of concurrent task execution and I/O processing.

I/O requests can be issued by user MACRO or FORTRAN programs by means of the functions described in this manual. All of these functions generate forms of the QUEUE I/O system directive. I/O functions can be used to request performance of such operations as opening and closing I/O files, reading and writing, obtaining information about a specified I/O device handler task, and giving a task exclusive use of an I/O device.

This manual describes everything a user must know in order to write programs that request I/O operations of standard I/O device handler tasks. Chapter 2 describes the operations of certain basic I/O functions available to all devices in the RSX system. Each subsequent chapter details the operation of one device and the handler task associated with that device, and describes I/O functions specific to that device.

1.2 I/O TABLES, LISTS AND TASKS

The following sections describe the basic tables, lists and tasks used in RSX I/O processing.

1.2.1 Logical Unit Table

I/O requests are made to a logical device identified by a logical unit number (LUN). Each LUN assigned in RSX corresponds, at least temporarily, to a particular physical device. LUN assignments can initially be made at system startup time and can subsequently be reassigned via the REASSIGN MCR function or the ASSIGN MULTIACCESS

Monitor command by setting up a Logical Unit Table (LUT) that relates logical unit numbers to the corresponding physical device assignments.

The Logical Unit Table Is a block of contiguous core with a one-word entry or slot for each possible logical unit number in RSX. Each entry consists of the address of a PDVL node. Slots are assigned sequentially from LUN-1 to LUN-64. (It is possible to change the number of LUNs in the system, up to a maximum of 511, by reassembling the RSX Executive.) LUT entries corresponding to unassigned LUNs contain zeros. When a LUN assignment is made, the LUT slot is filled with the address of a node in the Physical Device List (PDVL) that contains information about the physical device associated with the relevant LUN. For example, the physical device "terminal unit 1" may be assigned to logical unit 12 (LUN-12). The twelfth Logical Unit Table entry then contains the address of the PDVL node that has information on terminal unit 1.

In RSX systems with MULTIACCESS, there are two distinct types of LUNs: "system" LUNs and "virtual" LUNs. System LUNs are those logical unit numbers related to physical devices via the LUT. This is precisely the relationship described in the previous paragraphs. System LUNs are numbered in the same way that LUT entries are numbered (the first LUT entry - the entry with the lowest absolute address - corresponds to system LUN-1, and so forth. Virtual LUNs are assigned to users in blocks of 25 LUNs, always numbered from 1 to 25. Virtual LUNs are the only LUNs that a user task can access. The MULTIACCESS Monitor ensures that the virtual LUNs accessed by a user are "mapped" into system LUNs prior to the queuing of any I/O request.

In the following descriptions of I/O functions, the use of the term "LUN" should be understood as virtual LUN for tasks run under control of MULTIACCESS and as system LUN for all other tasks.

1.2.2 Attach Flag Table

The Attach Flag Table (AFT) is constructed with entries parallel to those in the Logical Unit Table. The AFT contains one-word entries, each of which corresponds to a single LUN. These entries are normally set to zero. When a user task requests that a physical device be attached to his task by specifying a particular LUN, the AFT entry corresponding to that LUN is filled with the address of a node in the System Task List (STL) that contains a description of the task requiring attachment.

1.2.3 Physical Device List

The Physical Device List (PDVL) is a standard system list or deque. This deque consists of a series of nodes, each of which contains the following information concerning one unit of a physical device:

Word	Contents
0	Forward pointer
1	Backward pointer
2	Device name (first half)
3	Device name (second half always 0)
4	Device attach flag
5	Device unit number
6	Device request queue (deque list-head) forward pointer
7	Device request queue (deque list-head) backward pointer
10	Trigger event variable address
11	Assign inhibit and files open flag

When a task requests that a unit be attached, word 4 in the PDVL node for that unit is filled with the address of the AFT entry that references the attaching task and corresponds to the LUN assigned to the attached physical device.

1.2.4 I/O Rundown Task and Queue

I/O rundown is defined as delaying the availability of a core partition until all transfers to and from that partition have stopped or have been allowed to complete. This procedure is performed when a task exits or is forced to exit and its transfers-pending count is nonzero. ●

A task can exit in one of the following ways:

1. Ordinary exit (STOP for FORTRAN or CAL (10) for MACRO)
2. Termination by the Executive because of memory-protect or similar violation
3. Termination by the console operator via the ABORT MCR Function task

Checking for the need for I/O rundown is performed in the EXIT code in the Executive.

When a task exits, its transfers-pending count is examined. If the count is nonzero, the node for that task is removed from the Active Task List and inserted by priority in a deque of I/O requests called the I/O Rundown Queue (IORDQ). The task is not officially deactivated, however, until completion of I/O rundown. To accomplish this, the resident task IORD (I/O Rundown) is triggered to process entries in the I/O Rundown Queue. ●

IORD performs I/O rundown for only one task at a time. It uses the UNMARK system directive to nullify any outstanding mark-time requests made by the given task. Then, if the transfers-pending count is still nonzero, it scans the Physical Device List for active I/O handlers. As long as this count remains nonzero, the next active handler is called to stop I/O for the indicated task. IORD places a priority-zero ABORT I/O request at the head of the I/O request deque in the first PDVL node for the device. Magtape, for example, may have up to eight PDVL nodes; one per physical tape unit. The first node does not necessarily correspond to device unit 0. It is simply first in the PDVL. Only one request is made per handler.

After this request has been queued, IORD sets bit 2 of the handler trigger event variable, declares a significant event, and then waits for the handler to signal completion by setting the IORD event variable.

When the entire PDVL has been scanned and the transfers-pending count is still nonzero, the system assumes that some handler is not functioning properly. IORD outputs the following message containing the task name and transfers-pending count via the Task Termination Notice List to LUN-3*:

```
TASK ABORTED; YET TRANSFERS PENDING COUNT = xxxxxx
```

Meanwhile, the count is set to zero, the task is deactivated and the task partition is freed for use by other tasks.

1.3 I/O DEVICE HANDLER TASKS

Nearly all I/O operations are executed in a device-independent manner by tasks called I/O device handler tasks. One device handler task is ordinarily associated with all units of a particualr physical device - for example, with all DECtape units.

I/O device handler tasks for standard Digital-supplied devices are provided as part of the standard RSX system. It is possible for the user to write his own handlers, as well. Part XI of this manual (Construction of Advanced Tasks) contains instructions for coding an I/O handler. A summary of I/O handler names is given in Table 1-1.

* For tasks running under MULTIACCESS, the message is output via the TDV Exit Queue (TDV.EQ) instead of the Task Termination Notice List. The output goes to the user terminal instead of to LUN-3.

Table 1-1
RSX Devices

Device Name	Device	Handler Task
TTn	Terminal	TTY
DTn	DECtape	DT....
MTn	Magtape	MT....
DK	Disk Driver	DSK
RF	Fixed-Head Disk	RF....
RPn	Disk Pack	RP....
RKn	Disk Cartridge	RK....
PR	Paper Tape Reader	PR....
PP	Paper Tape Punch	PP....
CD	Card Reader	CD....
CP	Card Punch	CP....
LP	Line Printer	LP....
AD	Analog-to-Digital Converter	AD....
AF	Automatic Flying Capacitor Scanner	AF....
UD	Universal Digital Controller	UD....
CC	System COMMON Communicator	CC....
VTn	Display	VT....
VWn	Writing Tablet	VW....
XY	XY Plotter	XY....

Although under RSX most I/O is processed via handler tasks, this method is a convention and not a system requirement. It is possible for the user to perform I/O directly. For example, if a user has a special I/O device that is accessed by only one task, I/O service can be coded as part of that task. This offers the advantage of bypassing all Executive functions except for interrupt connect and disconnect logic. It is also easier and more efficient to write the code for a Task-oriented I/O routine than to create a new I/O Device Handler Task. This approach, however, does not support standard I/O calls and does not facilitate the protection of I/O Rundown.

1.3.1 Handler Initialization

Each time a LUN is assigned to a physical device unit, the Handler Task associated with that unit is requested by the REASSIGN MCR Function Task. When in core, the Handler initializes itself by connecting to an interrupt line and by entering its Trigger Event Variable address in the appropriate word of the corresponding Physical Device List node (or nodes if the device can have more than one unit).

This informs the system that the Handler is ready to service I/O requests. Then it uses the WAITFOR System Directive to suspend Handler execution until its Trigger Event Variable is set.

1.3.2 I/O Request Queuing

Tasks make I/O requests by specifying one of the available forms of the QUEUE I/O Directive. All I/O forms or functions are described in detail in subsequent sections of this manual. Each I/O request references a specific LUN, which must have a corresponding nonzero entry in the Logical Unit Table. The RSX Executive expects to find in this LUT slot the address of the Physical Device List node containing information about the physical device assigned to the relevant LUN. This PDVL node contains, in turn, the Trigger Event Variable address of the I/O Device Handler Task associated with this device. The address is set when the Handler Task is initialized.

If the Executive determines that the relevant LUT slot contains a zero, the I/O request is rejected because the LUN has not been assigned to a physical device. If the Trigger Event Variable address in the relevant PDVL node is zero, the request is rejected because the Handler Task has not been properly initialized. If all necessary information is present, information from the CAL Parameter Block (CPB) of the I/O request is placed in a new node which is inserted in an I/O request queue for the appropriate device (there is a separate I/O request queue for each physical device unit). Pointers to this queue are found in words 6 and 7 of the PDVL node corresponding to the device. The queue is ordered by the priority of the Task issuing the I/O function; requests of equal priority are inserted in the order in which requests are made.

After a request has been queued, the Handler Task associated with the requested device is "triggered" by setting the Handler's Trigger Event Variable and declaring a Significant Event. If a Handler Task is triggered while it is servicing a previous request, the request will be ignored until the request being processed is complete.

1.3.3 Event Variables

When issuing an I/O function from a MACRO or FORTRAN Task, the user can specify an Event Variable in the I/O call itself. This variable is set to inform the user about the status of the I/O request and the success of the I/O operation. If the request cannot be queued, and the user has specified an Event Variable, one of the following is returned:

Event Variable	Meaning
-101	LUN out of range
-102	LUN not assigned to a physical device
-103	Nonresident or noninitialized I/O Device Handler Task
-777	Deque node unavailable (empty pool)

If the request is successfully queued, the requester's Event Variable is set to zero, indicating that the request is pending. Eventually, when the requested operation is attempted, the Event Variable is set either to a positive value (normally +1), indicating success, or to a negative number whose value pinpoints the reason for failure of the operation. Appendix A lists and describes returned Event Variables.

1.3.4 I/O Request Processing

I/O requests are handled by the appropriate I/O Device Handler Task. An individual request is dequeued (removed from the queue) by the Handler at software priority API-7 (Task Level). Requests are typically processed in order of Task priority by stripping the front node from the device's request queue. However, if a device has been attached, the Handler for that device will service only requests from the attaching Task until a DETACH is accepted. The fact that one Task has attached a device does not prevent another Task from queuing I/O requests to that device. Pending or current I/O requests can be aborted by means of the I/O Rundown Task, described above.

In general, all I/O Handlers make validity checks on arguments which reference core memory, but the checks are made only for I/O requests from USER-mode Tasks. Few checks are made on the arguments passed on by EXEC-mode Tasks.

1.3.5 Handler Exit

When a logical unit/physical device relationship is dissolved by means of the REASSIGN MCR Function Task so that no further Logical Unit entries are assigned to a Handler, the following takes place:

1. The assign inhibit flag (word 11, bit 0) of the relevant device's PDVL node is set.

2. The Trigger Event Variable address (word 10) of the PDVL node is cleared to prevent acceptance of further requests for the device.

3. A DISCONNECT & EXIT request is inserted at the end of the request queue for the device. This is the last request a Handler sees because it is inserted with a priority of 514, which is lower than that of any other Task.

The Handler Task services this request by performing the following:

1. Causes all system resources (e.g., nodes) held by the Task to be released.

2. Closes any open files.

3. Ensures that I/O under Handler control has terminated.

4. Disconnects from the device's interrupt line.

5. Clears the assign inhibit flag in the PDVL node associated with the Handler.

6. Exits.

1.3.6 I/O Data Modes

RSX utilizes four distinct data modes. These are shown below, with corresponding codes:

Table 1-2
I/O Data Modes

Code	I/O Data Mode
0	IOPS BINARY
1	IMAGE BINARY
2	IOPS ASCII
3	IMAGE ASCII

The implications and uses of these data modes are described in detail in the XVM/DOS USER'S MANUAL. Line-buffer construction under RSX is also the same as the description of logical records in the XVM/DOS manual.

The devices summarized in the following table support one or more of the standard I/O data modes.

Table 1-3
I/O Data Modes by Device

Handler Name	Device	I/O Data Mode
TTY	Terminal	IOPS ASCII IMAGE ASCII
DT....	DECtape	All modes
MT....	Magtape	All modes
RF....	Fixed-Head Disk	All modes
RP....	Disk Pack	All modes
RK....	Disk Cartridge	All modes
PR....	Paper Tape Reader	All modes
PP....	Paper Tape Punch	All modes
CD....	Card Reader	IOPS ASCII
LP....	Line Printer	IOPS ASCII IMAGE ASCII
XY....	XY Plotter	IOPS BINARY IOPS ASCII

Data modes are not applicable to the Disk Driver, the AD Analog-to-Digital Converter, the AFC Automatic Flying Capacitor Scanner, the UDC Universal Digital Controller, the COMMON Communicator, and certain other devices.

CHAPTER 2

STANDARD I/O FUNCTIONS

This chapter describes in some detail all of the basic I/O functions common to standard RSX devices. Each section describes one form of the QUEUE I/O System Directive. In all models and examples included in this chapter, the following conventions apply:

1. A space in the text indicates an actual space in the function call.

2. Square brackets ([ ]) indicate optional parameters.

3. In "Form" models, upper case characters indicate those required by the system, while lower case characters indicate entries which are to be specified by the user and are dependent on the particular Task.

Rules about line termination and error correction conform to MACRO or FORTRAN program standards.

Each of the I/O functions described in this Chapter has a unique function code. The following table summarizes codes and calls for all standard I/O calls.

Table 2-1
Standard I/O Functions

Code	MACRO Call	FORTRAN Call	Function
1700	---	---	ABORT I/O for Task
2300	---	---	PREALlocate I/O buffer space
2400	ATTACH	CALL ATTACH	Obtain exclusive use of I/O device
2500	DETACH	CALL DETACH	Release attached device
2600	READ	READ	READ input into buffer
2700	WRITE	WRITE	WRITE output from buffer
3200	SEEK	CALL SEEK	Open file for input
3300	ENTER	CALL ENTER	Open file for output
3400	CLOSE	CALL CLOSE	CLOSE file
13400	---	---	ERROR close file -- internal disk file Handler function
3500	DELETE	CALL DELETE	DELETE file name from directory
3600	HINF	CALL HINF	Request Handler Task information
3700	RENAME	CALL RENAME	Open file for RENAMing
77700	---	---	DISCONNECT from interrupt line & EXIT

2.1 QUEUE I/O: QUEUING REQUESTS FOR A DEVICE

The QUEUE I/O system directive is responsible for generating calls to a variety of I/O functions defined in RSX and described in subsequent sections of this chapter. This directive places an I/O request for a device in a queue of requests for a given unit on that device. In all cases, the inclusion of a logical unit number (LUN) in the I/O call serves to identify the device. However, a few I/O calls (e.g., ALLOCATE, DEALLOCATE, GET and PUT) are device-dependent and, when specifying one of these, the user must include the address of a device-dependent table of control information.

Prior to sending an I/O request to a device handler (i.e., queueing an I/O request), the QUEUE I/O directive determines whether the request was made by a task accessing a virtual (mapped) LUN. If so, and the LUN specified in the QUEUE I/O directive is between 1 and 25, the directive maps the virtual LUN into the corresponding system LUN as it queues the request.

The QUEUE I/O directive is implemented as a set of CAL parameter blocks (CPBs), each of which contains a unique function-code word. This word indicates the specific operation to be performed in connection with the requested device.

This chapter illustrates MACRO and FORTRAN calls for different forms of QUEUE I/O. The generalized CPB for this directive is:

Word	Contents
0	CAL function code (00; bits 12-17) and I/O function code (bits 3-11)
1	Event variable address
2	Logical unit number (LUN)*
3	Unique to I/O call
4	Unique to I/O call
5	Unique to I/O call

As is clear from the above, the I/O function code included in Table 2-1 actually consists of two components: a CAL function code of 00, indicating a form of QUEUE I/O, and an actual I/O function code preceding the CAL function code, indicating the particular form of QUEUE I/O that is generated. A directive code of 0000 is illegal.

As in all system directives, the event variable is filled with a code (Appendix A) indicating that an operation is pending, has succeeded or has failed. In the case of a failure, the code also indicates a reason. Pending status of a call can occur in the following way. Once the QUEUE I/O directive has accepted the call as valid, it enters

*LUNs in MACRO programs are usually coded in octal, but they are coded in decimal elsewhere in the system (e.g., FORTRAN code or REASSIGN MCR function).

the request in the device I/O queue, sets the trigger event variable for the device, declares a significant event, and then passes control to the handler. There is a potential delay during which the request is queued but has not been acted on by the I/O device handler. During this time, the event variable associated with the I/O request is set to zero (the "pending" state).

After issuing a call to execute some I/O function, the user task must generally issue a call to wait for function completion, which is signaled when the handler sets the associated event variable to a nonzero value. This wait is required only at the point where the user task cannot continue processing without assurance that the I/O function has completed. The exceptions to the preceding rule are a few specific FORTRAN statement-level I/O functions (e.g., READ, WRITE, REWIND and ENDFILE), which have no associated event variables. In these cases, the wait for completion is performed by the FORTRAN OTS routines responsible for issuing the call.

If the directive has been issued by a user task, the name and priority of the issuing task are used as the requester name and priority. If the directive is issued from an interrupt service routine, no name is assigned to the request and a default priority of one is supplied. Following is an example of an I/O call. This particular function reads input into a buffer. Examples of MACRO expansions for different I/O calls appear in subsequent sections of this chapter.

```
CPB  2600       /CAL PARAMETER BLOCK TO READ (I/O FUNCTION
                /CODE=26;  CAL FUNCTION CODE=00)
     EV         /ADDRESS OF EVENT VARIABLE
     4          /FROM LUN #4
     2          /IN IOPS ASCII (MODE 2).
     BUFFER     /TRANSFER TO CORE IS TO BEGIN AT
     100        /"BUFFER" AND CONTINUE FOR NO MORE THAN
                /100 (OCTAL) WORDS.
```

Wherever possible, it is recommended that the user specify an event variable when issuing I/O requests. The specification of an event variable is desirable because, under most circumstances, the user should be aware of any I/O errors that occur. It is the event variable that informs the user of these errors. Furthermore, the event variable is used to signal the completion of an I/O operation. In most cases, the WAITFOR directive is closely tied to the event variable. The directive is described in Part V of this manual.

2.2 ABORT: ABORTING I/O FOR A TASK

The ABORT function generates a form of the QUEUE I/O directive that is normally issued only by the I/O Rundown task, IORD. It informs the appropriate I/O device handler tasks that they must stop all I/O for the task to be ABORTed. ABORT functions cannot typically be issued by user tasks. Only the terminal handler, TTY, will process such a request if issued by a task other than IORD. The I/O function called ABORT should not be confused with the ABORT MCR Function task.

This function is inserted at the head of the queue with priority code zero; as though a task at priority zero has issued the QUEUE I/O directive. Also, whether or not the physical device is attached, the DQRQ subroutine in the Executive dequeues the ABORT request.

The ABORT CPB has the following format:

Word	Contents
0	1700 (I/O function code)
1	Event variable address
2	Logical unit number (zero)

2.3 ATTACH: ATTACHING AN I/O DEVICE

The ATTACH function generates a form of the QUEUE I/O System Directive which requests the exclusive use of an I/O device for the calling Task. If the Directive is accepted, no other Task can use the device, regardless of priority. However, all requests by other Tasks are queued. When the device is freed by a DETACH call, requests queued by other Tasks are once again considered on the basis of Task priority. Only the DETACH I/O function or the REASSIGN MCR function can override an ATTACH.

ATTACH functions can be issued by MACRO and FORTRAN programs. Following is the FORTRAN call:

Form:	CALL ATTACH(LUN[,ev])
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable
Example:	Request ATTACHment of the device assigned to LUN-32: CALL ATTACH(32,IEV)

The CAL Parameter Block (CPB) for this form of QUEUE I/O consists of the following:

Word	Contents
0	2400 (I/O function code)
1	Event Variable address
2	Logical Unit Number

Following is the MACRO expansion for this function:

```
/
/ **** ATTACH LUN[,EV]
/
        .DEFIN ATTACH,LUN,EV
        CAL    .+2
        JMP    .+4
        2400
        EV+0
        .DEC; LUN; .OCT
        .ENDM
```

2.4 CLOSE: CLOSING A FILE

The CLOSE function generates a form of the QUEUE I/O Directive which informs the I/O Device Handler Task assigned to the specified Logical Unit Number that the issuing Task has completed a set of related I/O operations on the current file. CLOSE is used primarily to directoried devices to close a file that was opened via SEEK, ENTER, or RENAME. It is also interpreted in a special way by the Paper Tape Reader Handler to mean unload the tape.

Because issuing a CLOSE implies that a file is currently open on the specified LUN, a file name is only needed as part of the call to specify a new file name following a RENAME. If the CLOSE is accepted, subsequent references to the CLOSEd file are not allowed until an appropriate SEEK, ENTER, or RENAME is issued.

CLOSE functions can be issued by MACRO and FORTRAN programs. Following is the FORTRAN call:

Form:	CALL CLOSE(LUN,[nHname,nHext[,ev]])
Where:	LUN is decimal and represents the Logical Unit Number n is the number of characters in name/ext name of file CLOSEd is a string of one to five ASCII characters ext represents the file extension and is a string of one to three ASCII characters ev is the integer Event Variable
Example:	CLOSE file named DATA SRC on the device associated with LUN-6: CALL CLOSE(6,4HDATA,3HSRC,IEV) or CALL CLOSE (6)

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	3400 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	File name (first half)
4	File name (second half)
5	File name extension

Words 3 through 5 of this CPB are required only when CLOSing a file which has been opened in order to be renamed. Otherwise, a three-word CPB is sufficient.

Following is the MACRO expansion for this function:

```
/
/ **** CLOSE   LUN[,FLNAM,EXT[,EV]]
/
       .DEFIN  CLOSE,LUN,FLNAM,EXT,EV
       CAL     .+2
       JMP     .+7
       3400
       EV+0
       .DEC ;LUN;  .OCT
..=.;  .SIXBT  'FLNAM'
       0;      .LOC      ..+2
       .SIXBT  'EXT'
       .ENDM
```

A special form of the CLOSE function, called ERROR CLOSE, exists primarily for internal use within the disk file Handlers.

2.5 DELETE: DELETING A FILE

The DELETE function generates a form of the QUEUE I/O Directive which requests the I/O Device Handler Task assigned to the specified Logical Unit Number to remove a particular file name from the device's file directory.

DELETE functions can be issued by MACRO and FORTRAN programs. Following is the FORTRAN call:

Form:	CALL DELETE(LUN,nHname,nHext[,ev])
Where:	LUN is decimal and represents the Logical Unit Number n is the number of characters in name/ext name of file DELETEd is a string of one to five ASCII characters ext represents the file extension and is a string of one to three ASCII characters ev is the integer Event Variable
Example:	DELETE file named DATA SRC from the directory of the device associated with LUN-6: CALL DELETE(6,4HDATA,3HSRC,IEV)

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	3500 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	File name (first half)
4	File name (second half)
5	File name extension

Following is the MACRO expansion for this function:

```
/
/ **** DELETE  LUN,FLNAM,EXT[,EV]
/
       .DEFIN  DELETE,LUN,FLNAM,EXT,EV
       CAL     .+2
       JMP     .+7
       3500
       EV+0
       .DEC;  LUN;  .OCT
..=.;  .SIXBT "FLNAM"
       0; .LOC ..+2
       .SIXBT "EXT"
       .ENDM
```

2.6 DETACH: FREEING AN ATTACHED I/O DEVICE

The DETACH function generates a form of the QUEUE I/O System Directive which releases an attached device from exclusive use by the calling Task. If the DETACH is accepted, previous requests which were queued while the device was attached can be processed. The Task issuing the DETACH must be the same Task which issued the ATTACH. A DETACH to an unattached device is treated as a valid request.

DETACH functions can be issued by MACRO and FORTRAN programs. Following is the FORTRAN call:

Form:	CALL DETACH (LUN[,ev])
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable
Example:	DETACH the device assigned to LUN-32: CALL DETACH(32,IEV)

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	2500 (I/O function code)
1	Event Variable address
2	Logical Unit Number

Following is the MACRO expansion for this function:

```
/
/ **** DETACH  LUN[,EV]
/
        .DEFIN  DETACH,LUN,EV
        CAL     .+2
        JMP     .+4
        2500
        EV+0
        .DEC;  LUN;  .OCT
        .ENDM
```

2.7 DISCONNECT & EXIT: CLEARING THE INTERRUPT LINE

The DISCONNECT & EXIT function is requested only by the REASSIGN MCR Function when no LUNs are associated with the given I/O device. Acceptance of this function causes the Handler to disconnect from its interrupt line, to clear flags in the Physical Device List (PDVL) mode for device unit zero, and to exit.

2.8 ENTER: OPENING A FILE FOR OUTPUT

The ENTER function generates a form of the QUEUE I/O Directive which opens a sequential-access file for output. It is not used for random-access files. ENTER requests the I/O Device Handler Task assigned to the specified Logical Unit Number to search the file directory of the associated device for a free directory entry in which to place a new file name. If a file of the same name already exists, it will be deleted and replaced when the new file is closed. For the case in which the device is DECtape, an entry for the new file is not recorded until the file is closed. For disk, however, an entry is made by the ENTER function prior to writing, because multiple Tasks may write files on the disk simultaneously. If the file is not already being manipulated, it is opened for output only. On directoried devices, ENTER must be issued before any data transfers using the WRITE function can be initiated.

ENTER functions can be issued by MACRO and FORTRAN programs. Following is the FORTRAN call:

Form:	CALL ENTER(LUN,nHname,nHext[,ev])
Where:	LUN is decimal and represents the Logical Unit Number n is the number of characters in name/ext name of file ENTERed is a string of one to five ASCII characters ext represents the file extension and is a string of one to three ASCII characters ev is the integer Event Variable
Example:	ENTER a file named DATA SRC into the directory of the device associated with LUN-6: CALL ENTER(6,4HDATA,3HSRC,IEV)

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	3300 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	File name (first half)
4	File name (second half)
5	File name extension

Following is the MACRO expansion for this function:

```
        /
        / **** ENTER   LUN,FLNAM,EXT[,EV]
        /
               .DEFIN  ENTER,LUN,FLNAM,EXT,EV
               CAL     .+2
               JMP     .+7
               3300
               EV+0
               .DEC;  LUN;  .OCT
        ..=.;  .SIXBT "FLNAM"
               0; .LOC ..+2
               .SIXBT "EXT"
               .ENDM
```

2.9 HINF: REQUESTING I/O HANDLER INFORMATION

The HINF function generates a form of the QUEUE I/O Directive which provides certain information about the physical device and I/O Device Handler Task associated with a specified Logical Unit Number. Handler information is coded into a single word which is stored in the Event Variable of the calling Task. The Event Variable is thus a required parameter in the HINF call. HINF functions can be issued by MACRO and FORTRAN programs. Following is the FORTRAN call:

Form:	CALL HINF(LUN,ev)
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable
Example:	Request information about the device and I/O Device Handler Task associated with LUN-6: CALL HINF(6,IEV)

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	3600 (I/O function code)
1	Event Variable address
2	Logical Unit Number

If HINF is accepted, the Event Variable is filled with the following device and Handler information:

Bit	Contents
0	Always zero; this implies that all actual Handler information is represented by a positive Event Variable; a negative Event Variable indicates that Handler information is not available (e.g., LUN is not assigned)
1	Input: set to 1 if data can be input from the device to a Task
2	Output: set to 1 if data can be output from a Task to the device
3	Directory-Oriented: set to 1 if the Handler accesses files on the device via a file directory; this implies that a file must be referenced by name and opened for input or output before reading or writing can be performed
4-11	Unit Number: must be in range 0-255 decimal
12-17	Device Code: must be in range 1-63 decimal; zero is illegal; see Table 2-2 for standard DEC device codes; users should assign codes to their own devices in reverse order, beginning with 63.

The following codes have been assigned to standard RSX devices:

Table 2-2
I/O Devices and Codes

Device Code	Handler Name	Device
1	TTY	Terminal
2	RF....	Fixed-Head Disk
3	RP....	Disk Pack
4	DT....	DECtape
5	MT....	Magtape
6	PR....	Paper Tape Reader
7	CD....	Card Reader (CR15 and CR03B)
10	PP....	Paper Tape Punch
11	LP....	Line Printer
12		Reserved For Storage Scope Handler
13	VT....	Display
14	CC....	System COMMON Communicator
16	AF....	Automatic Flying Capacitor Scanner
17	UD....	Universal Digital Controller
20	AD....	Analog-to-Digital Converter
21	BD....	Batch Handler Task
22	CP....	Card Punch
23	VW....	Writing Tablet
24	RK....	Disk Cartridge
25	CD....	Card Reader (CR11)
26	XY....	XY Plotter

Following is the MACRO expansion for this function:

```
        /
        / **** HINF    LUN,EV
        /
              .DEFIN  HINF,LUN,EV
              CAL     .+2
              JMP     .+4
              3600
              EV+0
              .DEC;  LUN;  .OCT
              .ENDM
```

2.10 PREAL: PREALLOCATING I/O BUFFERS

The PREAL function generates a form of the QUEUE I/O Directive which PREALlocates an area of a Task's partition for I/O buffers. This function reserves storage but does not allocate it to a specific device.

PREAL is typically not called by user Tasks. It is relevant to such system programs as the FORTRAN IV Compiler and the MACRO Assembler, which utilize "free core" to build dynamic tables. PREAL is used prior to and in conjunction with the RAISEB System Directive. Its CPB consists of the following:

Word	Contents
0	2300 (I/O function code)
1	Event Variable address
2	Logical Unit Number

2.11 READ: READING FROM AN I/O DEVICE

The READ function generates a form of the QUEUE I/O System Directive which READs input into a buffer via the I/O Device Handler Task assigned to the specified Logical Unit Number. Input may be entered in any of the data modes supported by RSX, including IOPS BINARY,(0) IMAGE BINARY,(1) IOPS ASCII,(2) and IMAGE ASCII.(3)

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	2600 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	I/O data mode
4	Core line-buffer starting address
5	Core line-buffer size

If the user wants to READ from within a FORTRAN task, no subroutine CALL is required. He can simply use the standard FORTRAN READ statement described in the FORTRAN IV XVM LANGUAGE MANUAL. Following is an example of a FORTRAN READ which will read data in IOPS ASCII from the device assigned to LUN-3 and store them in the buffer named TXTBF:

```
          DIMENSION TXTBF (128)
             .
             .
             .
          READ (3,10) TXTBF
    10    FORMAT(128A5)
```

Following is the MACRO expansion for this function:

```
/
/ **** READ     LUN,MODE,BUFF,SIZE[,EV]
/
       .DEFIN   READ,LUN,MODE,BUFF,SIZE,EV
       CAL      .+2
       JMP      .+7
       2600
       EV+0
       .DEC;  LUN;   .OCT
       MODE
       BUFF
       SIZE
       .ENDM
```

2.12 RENAME: RENAMING A FILE

The RENAME function generates a form of the QUEUE I/O Directive which opens a file for RENAMing. It requests the I/O Device Handler Task assigned to a specified Logical Unit Number to search the file directory of the associated device for a specified file name. The actual process of RENAMing this file is not completed until after the user issues a CLOSE function in which the new file name is given.

RENAME functions can be issued by MACRO and FORTRAN programs. Following is the FORTRAN call:

Form:	CALL RENAME(LUN,nHname,nHext[,ev])
Where:	LUN is decimal and represents the Logical Unit Number n is the number of characters in name/ext name of file to be RENAMEd is a string of one to five ASCII characters ext represents the file extension and is a string of one to three ASCII characters ev is the integer Event Variable

Example:	Search the directory of the device assigned to LUN-8 for a file named DATA 001 and rename the file to DATA 002:
	`CALL RENAME(8,4HDATA,3H001,IEV)`
	`CALL WAITFR(IEV)`
	`IF(IEV)10,10,20`
	`C    LABEL 10=ERROR LOCATION`
	`20   CALL CLOSE(8,4HDATA,3H002,IEV)`
	`CALL WAITFR(IEV)`

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	3700 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	File name (first half)
4	File name (second half)
5	File name extension

Following is the MACRO expansion for this function:

```
        /
        / ***   RENAME  LUN,FLNAM,EXT[,EV]
        /
                .DEFIN  RENAME,LUN,FLNAM,EXT,EV
                CAL     .+2
                JMP     .+7
                3700
                EV+0
                .DEC;  LUN;  .OCT
        ..=.;   .SIXBT "FLNAM"
                0; .LOC ..+2
                .SIXBT "EXT"
                .ENDM
```

2.13 SEEK: OPENING A FILE FOR INPUT

The SEEK function generates a form of the QUEUE I/O Directive which requests the I/O Device Handler Task assigned to the specified Logical Unit Number to search the file directory of the associated device for a specified file name. If the file exists and is not being manipulated in a conflicting way, it is opened for input only. SEEK pertains only to sequential-access files, not to random-access files. On directoried devices, SEEK must be issued before any data transfers using the READ function can be initiated.

SEEK functions can be issued by MACRO and FORTRAN programs. Following is the FORTRAN call:

Form:	CALL SEEK(LUN,nHname,nHext[,ev])
Where:	LUN is decimal and represents the Logical Unit Number n is the number of characters in name/ext name of file sought is a string of one to five ASCII characters ext represents the file extension and is a string of one to three ASCII characters ev is the integer Event Variable

Example:	Search directory of device assigned to LUN-6 for a file named DATA SRC:
	`      CALL SEEK(6,4HDATA,3HSRC,IEV)`
	`C     WAIT FOR SEEK TO COMPLETE`
	`      CALL WAITFR(IEV)`

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	3200 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	File name (first half)
4	File name (second half)
5	File name extension

Following is the MACRO expansion for this function:

```
        /
        / **** SEEK     LUN,FLNAM,EXT[,EV]
        /
               .DEFIN   SEEK,LUN,FLNAM,EXT,EV
               CAL      .+2
               JMP      .+7
               3200
               EV+0
               .DEC;  LUN;  .OCT
        ..=.;  .SIXBT "FLNAM"
               0; .LOC ..+2
               .SIXBT "EXT"
               .ENDM
```

2.14 WRITE: WRITING TO AN I/O DEVICE

The WRITE function generates a form of the QUEUE I/O Directive which WRITEs output from a buffer to the I/O Device Handler Task assigned to the specified Logical Unit Number. Output may be produced in any of data modes IOPS BINARY, IMAGE BINARY, IOPS ASCII, or IMAGE ASCII.
(0) (1) (2) (3)

The CPB for this form of QUEUE I/O consists of the following:

Word	Contents
0	2700 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	I/O data mode
4	Core line-buffer starting address

If the user wants to WRITE from within a FORTRAN task, no subroutine CALL is required. He can simply use the standard FORTRAN WRITE statement described in the FORTRAN IV XVM LANGUAGE MANUAL. Following is an example of a FORTRAN WRITE which will write IOPS ASCII data from the buffer named TXTBF to the device assigned to LUN-3:

```
         DIMENSION TXTBF(128)
            •
            •
            •
         WRITE (3,10)TXTBF
   10    FORMAT(1X,128A5)
```

The next example illustrates the WRITing of a message, followed by a frequency, on LUN-3:

```
         WRITE(3,10)IFREQ
   10    FORMAT(29H WARNING, XFC OSCILLATING AT ,I6//)
```

Following is the MACRO expansion for this function:

```
/ **** WRITE   LUN,MODE,BUFF[,EV]
/
       .DEFIN  WRITE,LUN,MODE,BUFF,EV
       CAL     .+2
       JMP     .+6
       2700
       EV+0
       .DEC;  LUN;  .OCT
       MODE
       BUFF
       .ENDM
```

CHAPTER 3

DISK FILE I/O AND RF DISK I/O

3.1 RELATIONSHIP OF RF, RP, AND RK

RSX has sequential and random-access disk file I/O Handler Tasks for three types of disk: RF Fixed-head DECdisk, RK cartridge disk, and RP disk pack. These Tasks should not be confused with the Disk Driver Task. The sources for these Tasks can be conditionally assembled to produce all three versions. Most of this chapter applies to all three, but to avoid cumbersome notation, this chapter refers only to the RF. Chapters on the RP and RK describe the exceptions.

RSX supports up to eight RP units, up to eight RK units, one RF with up to eight platters, or any combination of these. Only one disk can be the system device (the device on which the RSX core image resides). A user with a mixture of disks can choose any one of the following as system disk: RF, unit 0 of the RK's or unit 0 of the RP's.

Previous versions of RSX supported only the RF disk. References to RF disk platters and disk addresses have the same meaning as in the earlier versions, but these terms have slightly different meanings when applied to RP and RK disks. Similarly, RSX still supports word addressability for the RF, but the RP and RK are block-oriented.

3.2 RF HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for RF DECdisk operation is named RF.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is RF for purposes of reassignment.

3.3 DESIGN DECISIONS

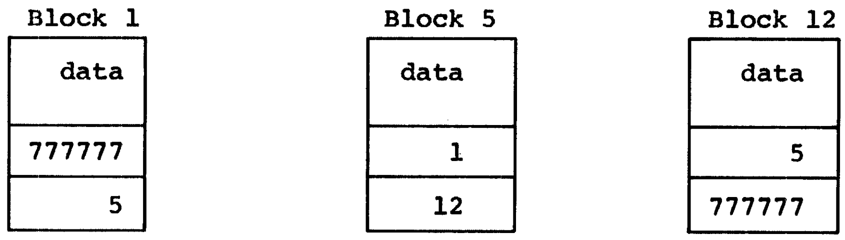
The RF Fixed-head DECdisk has two types of files: sequential access and random access.

Disk file Handler design allows DOS and RSX to reside on the same disk. The structures of bit maps, MFDs (Master File Directories), and UFDs (User File Directories) are identical. Sequential files in RSX are identical to those in DOS, but their random-access files differ.

Because the directory is open-ended, a virtually limitless number of sequential and random-access files may exist on the disk. In addition, any number of files may be simultaneously active or open.

3.4 SEQUENTIAL-ACCESS FILES

The Disk I/O Driver Task allocates blocks of disk space in increments of 256 decimal words. A file consists of one or more of these blocks linked together to form a chain. These linkage pointers make it unnecessary to locate the blocks adjacent to one another. Example:



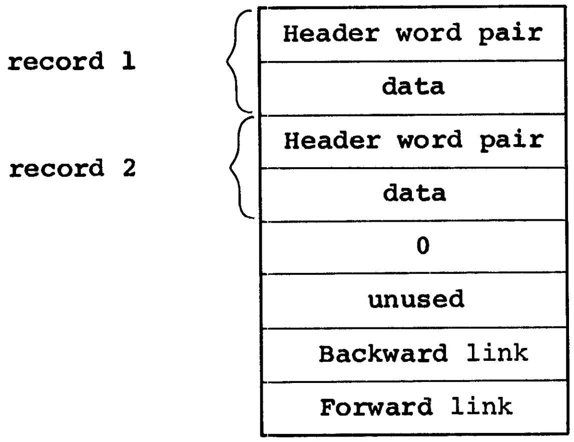
The figure above shows a three-block linked file. In each block, the first 254 (decimal) words may contain data. Word 255 is a backward link. It contains the number of the preceding file data block. If there is no preceding block, it contains 777777 (-1). Similarly, word 256 is a forward link, containing the number of the next file data block or 777777 if none follows.

The only way to determine where the nth file data block is on the disk is to read sequentially through the first n-1 blocks or look at the RIB (Retrieval Information Block(s) list). The XVM/DOS USER'S MANUAL describes the RIB in detail.

3.4.1 Data Records within a Sequential-Access File

Except for the restriction that physical data records must be 254 decimal words or less in length (in order to fit into a single disk block), the fact that sequential-access files consist of linked blocks should be logically transparent to user Tasks. (Physical records should not be confused with FORTRAN logical records.)

One or more data records may be stored within a disk block; however, data records may not be split between two blocks. The four data modes supported by this Handler all require line buffer header word pairs, which are recorded with the data. During creation (writing) of a file, if space remains in the current block, but the next record is too long to fit in the space, RF.... sets the next word to zero. A zero word in the position of a header indicates that no more data remains in the present block.



SAMPLE FILE BLOCK
(SEQUENTIAL-ACCESS FILE)

3.5 RANDOM-ACCESS FILES

At the level of the I/O Handler in DOS, random- and sequential-access files are both created as noncontiguous disk blocks. To perform DOS-style random access, the Retrieval Information Block (a list of the real disk location of each physical block in the file) must be in core. This can be very costly when simultaneous access to many random access files is needed.

Random-access files in RSX consist of contiguous disk storage. At the FORTRAN level, however, the user hardly notices the difference in recording techniques.

A single command to the I/O Handler can create a file. The file size is a multiple of the disk block size (256 words). The size may range from 1 to 777 octal blocks (one block less than half the capacity of one RF disk platter or 130,944 words).

Once the file has been created, the user must access the space allocated for it by calling on the resident Disk I/O Driver, not on this file Handler (RF). Calling on DSK speeds access to such files, because the file Handler processes only one I/O request at a time and could become tied up with lengthy file lookup requests. FORTRAN run-time subroutines automatically handle this process in a FORTRAN program.

Unlike sequential files, random-access files are only opened and closed to be renamed. They are simply created, and the user program has the responsibility of noting where the file is on the disk. FORTRAN run-time subroutines automatically handle this process in a FORTRAN program. Because the Disk I/O Driver handles access to the data in such files, there is no protection or interlock to keep two or more Tasks from trying to access the same data simultaneously. The danger, of course, is that one Task may modify part of the data while the other Task is trying to read it. Protection against this sort of problem is provided for sequential-access files. The cost, however, is that additional core is needed to remember the activity status of every open file and it takes longer to read/write data in sequential-access files.

A user Task normally does not have more than two or three sequential-access files open simultaneously because that requires additional buffer space within the Task partition of about 274 words (decimal) per file. On the other hand, the number of random-access files one may simultaneously access is potentially quite large because very little information about each file is needed. At the FORTRAN level, the maximum of four open files is governed by table space allotted in the DEFINE subroutine, part of the FORTRAN OTS library. The file capacity can be expanded by redefining an assembly parameter in DEFINE.

3.5.1 Data Records within a Random-Access File

Because the space within a random-access file on the disk is contiguous and because access to the data in such a file is handled by GET and PUT commands to the Disk I/O Driver, there is no such thing as a data record as far as the RF Handler is concerned. The user Task is free to read/write starting at any word in the file and may specify any word count (record size). At the FORTRAN level, the concept of a random-access data record takes on meaning because a structure is imposed by the FORTRAN run-time subroutines that handle random-access I/O.

3.6 USER FILE DIRECTORY (UFD)

The existence of each file is recorded in a file directory. It has an entry for each file giving its name, the number of its first block, the total number of blocks in the file, and the date it was created. For random-access files, two additional pieces of "accounting information" are recorded. When used by FORTRAN, the accounting information indicates the number of records in the file, the record size, and the type of data (formatted or unformatted).

To read data from a sequential-access file, a Task must first open it by issuing a SEEK command (from a MACRO program) or a CALL SEEK (from a FORTRAN program) so that the starting block may be found. To read data from a random-access file, a Task must issue a CREATE command (from a MACRO program) or CALL DEFINE (from a FORTRAN program) to find the location for the file or to create space if it does not exist.

File directories are called UFDs because they are identical to User File Directories in DOS. RSX supports multiple UFDs for each disk on the system. The LUN-UFD table in the executive is used to store the SIXBT representation of UFD names. This table has a one to one relationship with the LUT so a unique UFD can be specified for each LUN associated with a disk. Initially, during System Configuration, the LUN-UFD table is set to zero to indicate that no LUN's or UFD's are associated with any disk. This table may be subsequently modified in two ways. For "user" disks, the MNT MCR and TDV functions will enter the specified UFD name into the LUN-UFD table entries for all LUN's associated with the disk named. Furthermore, MNT will set the UFD name into the appropriate entry of the DISK-UFD table for future possible use by REASSIGN. The second way to modify the LUN-UFD table is via REASSIGN. When any disk is to be associated with a LUN, the user can specify a UFD name in REASSIGN's command string. If no such specification is made, REASSIGN will look up the default UFD name for the disk in the DISK-UFD table. This table has one entry for each disk which might be present on the system. Initially, the DISK-UFD table is zero except for the entry corresponding to the system disk. This entry is set to the .SIXBT representation of the RSX UFD by the System Configurator. When REASSIGN has obtained the UFD name, it enters the name into all entries of the LUN-UFD table for those LUN's being assigned to the disk specified. Hence, for each disk there can be as many UFD's accessible to a user's tasks as there are LUN's assigned to that disk. Whenever a UFD for a particular LUN disk pair must be accessed by the file handler, the handler obtains the name of the UFD associated with the LUN from the appropriate entry of the LUN-UFD table. The handler then scans the Master File Directory (MFD) of the disk to obtain the starting block of the UFD. Subsequently, the file handler can manipulate UFD entries with the data obtained from the MFD.

Each file entry consists of eight words and each UFD block can have 37 (octal) entries. The UFD can consist of several blocks linked in the same manner as a sequential-access file. The UFD is initially created by the System Configuration Task and is initially one block long and empty. Each UFD file entry has the following format:

Word	Contents
0	Filename (first half)
1	Filename (second half)
2	File extension
3	Truncation and first data block
4	Type and number of data blocks
5	First RIB block or accounting information
6	Protection code and pointer to start of RIB or accounting information
7	File creation date

Words 0-2, filename and extension, are entered in .SIXBT form. If the entry is unused, these words are zero. Word 3 consists of the following:

Bit	Contents
0	Set 1 if file is truncated
1-17	Number of file's first data block

Word 4 consists of the following:

Bit	Contents
0	File type (0 for sequential access; 1 for random access)
1-17	Number of data blocks in the file

When a sequential file is being created, an entry is made in the UFD (prior to any WRITE statements) so that a UFD slot is reserved. At this time the starting block number and the total file length (words 3 and 4) are unknown. The file cannot be read in this state. Bit 0 of word 3 is set to one to signal this, normally implying that the file is still being created. If, due to software or hardware error, file creation is never properly terminated, the file may remain in this unfinished, "truncated" state.

For sequential files, Word 5 contains the starting block having RIB data. Word 6 has a pointer for the first RIB block, indicating where the RIB data start. In bits 0-2, this word also has the protection code (set 1 for files written in RSX), though RSX ignores the code.

For RSX created files, Words 5 and 6 contain random access accounting information. Refer to the discussion on CREATE to obtain the format of these words.

Word 7 contains the date the file was created, using the following format:

Bit	Contents
0-5	Month (1-12, decimal)
6-11	Day (1-31, decimal)
12-17	Year (beginning at 1970 = 0, decimal)

3.7 RF HANDLER STRUCTURE

The RF Handler is written in several overlay seqments:

Resident Code	Task Initialization
	I/O function dequeue and dispatch
	HINF
	PREAL
	DISCONNECT & EXIT
	Block allocate and deallocate subroutines
	I/O request completed code
	Active LUN Deque
Overlay 1	Directives to open files:
	SEEK
	ENTER
	RENAME
Overlay 2	READ
	WRITE
Overlay 3	CLOSE
	ERROR CLOSE
	ABORT
Overlay 4	DELETE
Overlay 5	CREATE

I/O requests are processed to completion one at a time and are dequeued (removed from the queue) in the resident code. Provided that the I/O function is legal and that it was requested from Task level (this specifically excludes interrupt service routines), the dispatcher transfers to the appropriate routine, which might call in a different overlay from the disk.

The block allocate and deallocate subroutines are used in common by the overlays to get or relinquish disk space.

The I/O request completed code is a common exit point for all I/O functions when they have been completed.

The Active LUN Deque is a list maintained by RF.... to indicate which Tasks have open files and via which LUNs. More than one Task may open files via the same LUN, but each Task may have only one open file per LUN.

As a general rule, whenever an error occurs while a request is being processed and the requesting Task has a file open on the indicated LUN, the file is CLOSEd. An exception to this occurs when an already existing sequential file is in the process of being recreated. If an error occurs, the partially written new file is discarded and the old file is retained.

This could happen if a USER-mode Task were to EXIT immediately after a CLOSE without waiting for the CLOSE to finish.

3.8 RF HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the RF I/O Device Handler Task. Following are the legal function calls which may be issued for RF service and a brief description of each:

Table 3-1
RF Disk I/O Functions

RF Call	Function
HINF	Request Handler Task information
SEEK	Open file for input
ENTER	Open file for output
DELETE	Delete file name from directory
RENAME	Open file for renaming
PREAL	Preallocate I/O buffer space
ABORT	Abort I/O for a Task
DISCONNECT & EXIT	Disconnect Handler from interrupt line and exit
READ	Read input into buffer
WRITE	Write output from buffer
CLOSE	Close file
ERROR CLOSE	Handler internally closes file
CREATE	Create random-access file

Except For READ, WRITE, CLOSE, ERROR CLOSE, and CREATE, all the functions listed above operate almost exactly as described in Chapter 2. When requested by the RF Handler Task, however, these functions do require the few special considerations listed below.

Of the I/O functions listed above, some may be used any time. Others are legal only if they appear in the proper context or sequence, as shown below. Bracketed items [] may appear any number of times, including zero.

SEEK, [READ], CLOSE

ENTER, [WRITE], CLOSE

RENAME, CLOSE

DELETE (legal any time)

CREATE (legal any time)

HINF (legal any time)

PREAL (legal any time)

3.8.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +340002 to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 3 to indicate input and output device
3	Set 1 to indicate a directory-oriented Handler
4-11	Device unit 0
12-17	Device code 2 (RF disk)

3.8.2 SEEK: Opening a File for Input

The SEEK function opens a sequential-access file for input. SEEK must be executed prior to any READ statement. A file may be opened for input provided that it exists in the disk's file directory and provided that it is not open for any reason other than to be read. For instance, one cannot read a file if it is in the process of being rewritten or renamed.

If the SEEK is successfully performed, a node is added to the Active LUN Deque indicating that this Task has a file open on the specified LUN, and an I/O buffer is created in the top of the Task's partition to serve as an intermediate file data buffer. The Task must be in a partition large enough to contain a buffer for each file that is open at a given time.

3.8.3 ENTER: Opening a File for Output

The ENTER function opens a sequential-access file for output.

If the file does not exist, it is created by the ENTER-WRITE-...-CLOSE sequence. If the file already exists, ENTER opens the file to be rewritten. This function must be executed prior to any WRITE statement.

A file may be opened for output provided that it exists in the disk's file directory and provided that it is not already open.

As with SEEK and RENAME, when ENTER is successfully performed, a node is added to the Active LUN Deque to signal an open file and a file buffer is created in the partition of the requesting Task. The Task must be in a partition large enough to contain a buffer for each file that is open at a given time.

3.8.4 DELETE: Deleting a File Name from the Directory

The DELETE function is legal provided that the file exists in the directory and provided that it is not open (for reading, writing, or renaming).

3.8.5 RENAME: Opening a File for Renaming

The RENAME function opens a file for renaming. The file opened by RENAME is renamed by the CLOSE function. RENAME is identical to SEEK except that the file cannot be read and the file cannot be renamed if it is already open. Both sequential and random-access files can be renamed in this way.

3.8.6 PREAL: Preallocating I/O Buffer Space

The PREAL function preallocates space in the Task's partition for an I/O buffer.

This function is typically used by Tasks such as the Assembler and Compiler, which determine how much free core is available by issuing the RAISEB Directive. Because RAISEB increases the Task size, I/O buffers must first be reserved. PREAL would be unnecessary if the Task were to open all its files before issuing RAISEB. The buffer size used by RF.... is 274 decimal.

3.8.7 ABORT: Aborting I/O for a Task

ABORT closes all open files for a given Task and causes system resources (I/O buffers and nodes) to be relinquished.

3.8.8 DISCONNECT & EXIT: Clearing the Interrupt Line

The DISCONNECT & EXIT function requests RF.... to disconnect from its interrupt line (although RF.... is not connected) and to exit.

If any files are open, REASSIGN informs the operator via the MCR terminal (LUN-3) and asks if he wants to reassign anyway. If so, all open files are closed. As with ERROR CLOSE, however, partially rewritten files are deallocated so that the old files remain intact.

3.9 CLOSE: CLOSING A FILE

The CLOSE function is legal provided that a file has been opened. It operates almost exactly as described in Chapter 2, with the exceptions and special features described below.

If the file was opened by SEEK, CLOSing the file simply means relinquishing the I/O buffer (in the Task's partition) and removing the Active LUN node.

If the file was opened by ENTER, an end-of-file record is written and the file's entry in the file directory is updated to indicate the file's starting block and size. Then RIB blocks (Retrieval Information Blocks) are recorded for the file and the RIB data is recorded in the directory. If the file was being rewritten, the old version is removed from the directory and its blocks are deallocated. Then the I/O buffer and the Active LUN nodes are released.

If a file is being rewritten and the user Task neglects to wait for completion of the CLOSE function before it exits, e.g.,

```
        CALL  CLOSE (LUN, 4HFILE, 3HSRC)

        STOP
```

the old file may be retained and the new file discarded. The correct sequence is

```
        CALL  CLOSE (LUN, 4HFILE, 3HSRC, IEV)

        CALL  WAITFR (IEV)

        STOP
```

If the file was opened by RENAME, then CLOSE specifies the new file name. The CLOSE is legal provided that the new file is not being manipulated (as determined by a scan of the Active LUN Deque) and does not already exist in the directory. After the file is renamed, the I/O buffer and Active LUN node are released.

3.10 CREATE: CREATING A RANDOM-ACCESS FILE

The CREATE function CREATEs a random-access file, if it does not already exist, by allocating a contiguous block of disk storage and recording the file's existence in the file directory. The CAL Parameter Block (CPB) for CREATE is as follows:

Word	Contents
0	1600 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table consists of the following:

Word	Contents
0	Filename (first half)
1	Filename (second half)
2	File extension
3	File size (number of 256-word blocks)
4	Accounting information
5	Accounting information
6	Update indicator (zero for no update; nonzero for update)
7	Platter number (bits 0-2 contain unit number)
8	Disk address

Words 0-2, file name and extension, are entered in .SIXBT form.

The legal range for the file size is 1 to 777 octal blocks. If the file already exists, the size in the Control Table must match that of the existing file. If not, an error is declared (see below).

The accounting information need bear no relation to the file size. These two words are recorded as part of the file entry information in the directory when the file is first created. If the file already exists and the update indicator is zero, signifying that no update is requested, the accounting information in the CTB must match that in the file directory or else an error is declared (see below). If the update indicator is nonzero, the new accounting information is written into the directory entry. An example of the use of updating occurs when a user wishes to keep track of the last time he accessed the file. Thus, every time he accesses the file with a CREATE he must store a new date in the accounting information.

Because the accounting information is used by the random-access run-time subroutines, it is not accessible to the FORTRAN user. The first word (5) indicates the number of records in the file. The second word indicates the record size and the data mode. It has the following format:

Bit	Contents
0	Data mode (0 for unformatted (IOPS BINARY); 1 for formatted (IOPS ASCII))
1-17	Record size (words per record for IOPS BINARY; characters per record for IOPS ASCII)

If the file already exists and a mismatch error is detected, the correct file size and accounting information are stored in the CREATEr's Control Table. Regardless of error, if the file exists, then the starting location of the file (platter number and disk address) is returned to the Control Table. If the file did not previously exist, the starting location of the file is returned in words 7 and 8 of the Control Table. In this case, the values returned bear no relationship to the initial values of these two words. The user needs the starting location of the file so that he can construct GET and PUT requests to the Disk I/O Driver (DK via LUN-1). The FORTRAN user, however, does not need to know this. Once he has created the file with a CALL DEFINE statement, he employs the random-access form of the READ and WRITE statements as indicated in the FORTRAN manual. It is the random-access run-time subroutines that actually perform GET and PUT to the disk.

The CALL DEFINE statement is basically like the FORTRAN IV CALL DEFINE. The RSX version of this statement has three noteworthy differences.

```
CALL DEFINE (d,rsiz,fsiz,fnam,v,mode,a,dcod[,ev])
```

First, the argument shown as a is ignored in RSX (though something must appear in the calling sequence). This is the file size adjustment parameter. Under DOS-15, random-access files, once created, can be extended at a later time to a larger size. This is not so in RSX because of the requirement that storage in such a file be contiguous.

Second, when a random-access file is first created under DOS, the data in the file are initialized to zeros, because the file is created by sequential WRITEs. In RSX, the initial state of the data in the file is random.

Third, the RSX call has an additional parameter. It is an integer Event Variable, shown in brackets because it is optional. If the arguments in the call appear to be legitimate to the DEFINE subroutine, it attempts to create or look up the named file by calling on the disk file Handler. If the file Handler returns an error indication, this is returned in the Event Variable, if one was specified in the CALL DEFINE. Therefore, the user program should wait for the setting of the Event Variable by using CALL WAITFR. Following that, it should test for an error (negative Event Variable value) or success (positive Event Variable value).

+1 - file was already on disk

+2 - file did not previously exist

3.11 ERROR CLOSE: HANDLER'S INTERNAL CLOSING OF A FILE

The ERROR CLOSE function exists primarily for internal use within the RF file Handler. Whenever an I/O error is detected, RF puts an ERROR CLOSE request in its I/O queue as if it had been issued by the Task which caused the error. ERROR CLOSE is identical to CLOSE except when an already existing file is being recreated. In this case, the new file is discarded and the old file is retained. The new file's blocks are deallocated and its entry in the file directory is removed. If no old file exists, the new file is retained.

3.12 READ: READING INPUT INTO A BUFFER

The READ function READs input into a buffer via the RF Handler Task assigned to the specified Logical Unit Number. It operates almost exactly as in the description of READ in Chapter 2, with the exceptions and special features described below.

Provided that a file was opened with a SEEK, READ is legal in IOPS BINARY (Data Mode 0), IMAGE BINARY (Data Mode 1), IOPS ASCII (Data Mode 2), and IMAGE ASCII (Data Mode 3). All four modes require a line buffer header. Since the Handler does not have to pack or unpack data, it handles them in the same manner.

With every READ statement a data line is transferred to the requester's line buffer. The requesting Task should examine the header word to see if end-of-file has been reached or if a checksum, parity, or buffer overflow error has occurred. If READs are issued after end-of-file is reached, they result in the return of end-of-file.

Since several data lines fit into a disk block, not every READ requires a disk transfer.

If a checksum or buffer overflow error occurs, it applies only to a single data line. When a parity error occurs, however, it is flagged in each data line coming from the same file data block, since it is not known where in the block the error occurred.

3.13 WRITE: WRITING OUTPUT FROM A BUFFER

The WRITE function WRITEs output from a buffer to the RF Handler Task assigned to the specified Logical Unit Number. It operates almost exactly as in the description of WRITE in Chapter 2, with the exceptions and special features described below.

Provided that a file was opened with an ENTER, WRITE is legal in IOPS BINARY (Data Mode 0), IMAGE BINARY (Data Mode 1), IOPS ASCII (Data Mode 2), and IMAGE ASCII (Data Mode 3). All four modes require a line buffer header. Since the Handler does not have to pack or unpack data, it handles them in the same manner.

If the data line fits into the unused space in the current data block, it is stored there (in core) and WRITE processing is completed. If the line does not fit, the current block is written to the disk after another block is allocated. The new block then becomes the current block, and the data line is transferred to it.

Every data line has a two-word header at the beginning. The RF Handler sets the data mode (header word 0) and the checksum (header word 1).

CHAPTER 4

RK DISK I/O

This chapter describes the ways in which input/output operations for the RK cartridge disk differ from I/O for the RF DECdisk.

4.1 RK HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for RK disk operation is named RK.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is RKn for purposes of reassignment. The device code is 24 for uses in functions such as HINF.

The PDVL must have a node for each RK unit. Requests are dequeued on a priority basis, beginning with unit 0 and proceeding through the highest unit.

4.2 DISK PLATTERS AND DISK ADDRESSES

For the RK disk, the address and platter mentioned in Chapter 3 represent a 26-bit disk address. The high-order three bits of the platter word are reserved for a unit number when multiple RKs are present. The low-order eight bits of the platter word prefix the main disk address word, resulting in a 26-bit disk address.

The RK is block-oriented. Because disk accesses must therefore start on block boundaries, the low-order eight bits of the address word must be zero. Nevertheless, it is possible to access part of a block if access begins on a block boundary, but the rest of the block is zeroed, when writing only part of a block in this way. If access does not begin on a block boundary, an error occurs.

CHAPTER 5

RP DISK I/O

This chapter describes the ways in which input/output operations for the RP disk pack differ from I/O for the RF DECdisk.

5.1 RP HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for RP disk operation is named RP.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is RPn for purposes of reassignment. The device code is 3 for use in functions such as HINF.

The PDVL must have a node for each RP unit. Requests are dequeued on a priority basis, beginning with unit 0 and proceeding through the highest unit.

5.2 DISK PLATTERS AND DISK ADDRESSES

For the RP disk, the address and platter mentioned in Chapter 3 represent a 26-bit disk address. The high-order three bits of the platter word are reserved for a unit number when multiple RPs are present. The low-order eight bits of the platter word prefix the main disk address word, resulting in a 26-bit disk address.

The RP is block-oriented. Because disk accesses must therefore start on block boundaries, the low-order eight bits of the address word must be zero. Nevertheless, it is possible to access part of a block if access begins on a block boundary, but the rest of the block is zeroed, when writing only part of a block in this way. If access does not begin on a block boundary, an error occurs.

CHAPTER 6

DISK DRIVER I/O

The I/O Driver for the RF DECdisk, the RK cartridge disk, and the RP disk pack consists of two Tasks, DSK and DSA. DSK and DSA are permanently core-resident and form an integral part of the RSX Executive.

6.1 DISK DRIVER TASKS

These Tasks are considered an I/O Driver rather than an I/O Handler because they perform only primitive I/O functions and do not deal with file structuring and data modes. The RF, RK, and RP Handlers deal with the more complex functions.

Recorded on each disk is a series of bit maps which indicate those areas of the disk that are free (to be allocated) and those which are reserved (already allocated). Since there may be many bit maps per disk, it is possible for an ALLOCATE request to require several disk transfers (requiring up to 300 milliseconds). To prevent holding off high-priority disk requests for this length of time, the Disk Driver consists of two Tasks: DSK, the actual Disk Driver, and DSA, a lower-priority Task that handles disk ALLOCATE/DEALLOCATE requests.

Whenever DSK encounters an ALLOCATE or DEALLOCATE request in its I/O request queue, it simply moves the request node from its own queue to another queue belonging to DSA. Then it sets DSA's Trigger Event Variable and declares a Significant Event so that DSA will run (DSA runs at a priority level lower than DSK).

Previous versions of RSX supported only the RF disk. References to the RF disk platters and disk addresses have the same meaning as in the earlier versions, but these terms have slightly different meanings when applied to RP and RK disks. For these two disk types, the address and platter represent a 26-bit disk address. The high-order three bits of the platter word are reserved for a unit number when multiple RK's or RP's are present. The low-order eight bits of the platter word prefix the main disk address word, resulting in a 26-bit disk address.

The following calculation may be used to obtain a disk block number from the address and platter words when referring to any of the three types of disk:

```
        CLL
        LAC    ADDRESS
        LMQ
        LAC    PLATTER          /Remove unit number
        AND    (077777)
        LRSS   10               /AC now contains the
        LACQ                    /block number
```

The following calculation may be used to obtain a disk address and platter from a block number on any of the three types of disk:

```
    ECLA=641000                 /Clear AC with EAE
        LAC    BLOCK
        LMQ
        LLSS!ECLA 10
        DAC    PLATTER
        LACQ
        DAC    ADDRESS
```

Word addressability is still supported for the RF. The RK and RP, however, are block-oriented. Because disk accesses must therefore start on block boundaries, the low-order eight bits of the address word must be zero. Nevertheless, it is possible to access part of a block if access begins on a block boundary, but the rest of the block is zeroed, when writing only part of a block in this way. If access does not begin on a block boundary, an error occurs.

Only one disk can be the system device (the device on which DOS and RSX core image reside). A user with a mixture of disks can choose any one of the following as system disk: RF, unit 0 of the RK's, or unit 0 of the RP's.

6.2 BIT MAPS

Bit maps are identical to the SAT blocks in DOS. The location for the first bit map on each type of disk is:

Disk Type	Block Number
RF	1776
RK	1776
RP	764

The structure of these bit maps is as follows:

Word	Contents
0	Total number of blocks on the disk
1	Total number of blocks in this bit map
2	Number of blocks occupied on this bit map
3	First actual word of bit map
.	
.	
.	
376	Pointer to previous bit map or -1
377	Pointer to next bit map or -1

6.3 DISK ERRORS

Whenever a hardware-detected error occurs during a disk transfer, DSK reexecutes the transfer. If the error persists after the eighth try (the number of tries is actually an assembly parameter), a disk failure is assumed and the Event Variable associated with the I/O request is set with the contents of the Disk Status Register to help pinpoint the cause of the error.

Because not all Tasks can be expected to report Disk errors, two SCOM registers are updated by the Disk Driver:

Octal Register	Contents
152	The number of times disk transfers were retried
153	The number of disk failures that have occurred

6.4 DISK DRIVER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests for the Disk I/O Driver Tasks. Following are the legal function calls that may be issued and a brief description of each:

Table 6-1
Disk Driver I/O Functions

Driver Call	Function
HINF	Request Driver Task information
ABORT	Abort I/O for a Task
ALLOCATE	Reserve storage
DEALLOCATE	Free storage
GET	Read from disk
PUT	Write to disk

In addition, a FORTRAN utility routine called UPKEV has been implemented to aid in unpacking the Event Variable returned by the HINF function. All functions and subroutines are described in subsequent sections.

HINF and ABORT operate almost exactly as described in Chapter 2. When requested by the Disk I/O Driver, however, these two functions do require the few special considerations discussed below.

6.4.1 HINF: Requesting Driver Task Information

If a request for HINF is accepted, the Event Variable is set to +3000xx to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 3 to indicate input and output device
3	Set 0 to indicate a non-directory-oriented Driver
4-11	Device unit 0
12-17	Set xx, where xx is device code for the system disk (02 for RF; 03 for RP; 24 for RK)

6.4.2 ABORT: Aborting I/O for a Task

The Disk I/O Driver honors this function by removing from its queue all I/O requests made by the Task being ABORTed. Because the disk is a fast device, no attempt is made to stop an I/O transfer already in progress. The ABORT request is simply dequeued (removed from the queue) after the current transfer is completed.

6.5 ALLOCATE: RESERVING STORAGE

The ALLOCATE function generates a device-dependent form of the QUEUE I/O Directive which reserves storage on a disk. If the amount requested is available, information about the storage area is returned to a Control Table whose location is specified in the I/O call.

The information from the Control Table is not queued along with the information from the CAL Parameter Block. Therefore, after making the I/O request, one must not modify the Control Table until after completion of the request.

ALLOCATE functions can be issued only in EXEC mode, by both MACRO and FORTRAN programs. As explained above, the RF, RP, or RK may serve as system disk or "user" disk. Whether a disk is dedicated to the system or the user, the MACRO and FORTRAN calls it requires for ALLOCATion do not vary. The CAL Parameter Block and Control Table, however, do vary. Following is the FORTRAN call:

Form:	CALL DSKAL(table,words,dvcode,unit [,ev])
Where:	table is an integer array of at least three words representing the Control Table; a seven-word table is preferable if DSKGETs or DSKPUTs follow words is a decimal integer representing the number of words of storage desired dvcode is an integer representing the code of the device on which space will be allocated unit is an integer representing the unit number of the device on which space will be allocated ev is the integer Event Variable
Example:	ALLOCATE 768 words of disk storage on an RP disk, unit 2, and suspend execution until completion: DIMENSION ICTB (7) . . . IRP=3 IUNIT=2 CALL DSKAL (ICTB,768,IRP,IUNIT,IDKEV) CALL WAITFR (IDKEV)

For the system disk, the CPB for ALLOCATE consists of the following:

Word	Contents
0	1500 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

When an ALLOCATE request is issued for space on the system disk, word 0 of the Control Table is set with the amount of storage space desired:

Word	Contents
0	Requested storage (number of words)
1	Any value
2	Any value

Disk space allocation cannot occur immediately; therefore, it is necessary to specify an Event Variable address in the ALLOCATE CPB and to wait for completion of the request before attempting to read the parameters. If the ALLOCATE request for the system disk is honored, the 3-word Control Table is modified as follows:

Word	Contents
0	Storage granted (number of words)
1	Disk platter
2	Disk address

Because storage is allotted in blocks of 256 words (400 octal), the amount of storage granted may exceed the amount requested. The disk platter and disk address serve to locate the beginning of the storage block, which resides entirely on a single bit map.

130816 (377400 octal) is the upper limit on allowable allocation because 377401 would have to be rounded up to 400000, which is a negative number.

To reference any disk, the CPB for ALLOCATE consists of the following:

Word	Contents
0	11500 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address
4	Disk type (2 for RF; 3 for RP; 24 for RK; 0 for system disk)

When a generalized ALLOCATE request is issued, word 0 of the Control Table is set with the amount of space desired:

Word	Contents
0	Requested storage (number of words)
1	Device unit number
2	Any value

Word 1 of the Control Table has the following form:

Bit	Contents
0-2	Device unit number
3-17	Any value

If the ALLOCATE is honored, the Control Table is modified as follows:

Word	Contents
0	Storage granted (number of words)
1	Unit and platter numbers
2	Disk address

Word 1 of the Control Table then has the following form:

Bit	Contents
0-2	Device unit number
3-17	Platter number

6.6 DEALLOCATE: FREEING STORAGE

The DEALLOCATE function generates a device-dependent form of the QUEUE I/O Directive which frees a block of disk storage that was reserved by an ALLOCATE command.

The information from the Control Table is not queued along with the information from the CAL Parameter Block. Therefore, after making the I/O request, one must not modify the Control Table until after completion of the request.

DEALLOCATE functions can be issued only in EXEC mode, by both MACRO and FORTRAN programs. As explained above, the RF, RP, or RK may serve as system disk or "user" disk. Whether a disk is dedicated to the system or the user, the MACRO and FORTRAN calls it requires for DEALLOCATion do not vary. The CAL Parameter Block and Control Table, however, do vary. Following is the FORTRAN call:

Form:	CALL DSKDAL (table,dvcode,unit[,ev])
Where:	table is an integer array of at least three words, representing the Control Table; a seven-word table is preferable if ALLOCATEs, DSKGETs, or DSKPUTs use the same CTB dvcode is an integer representing the code of the device on which space is allocated unit is an integer representing the unit number of the device on which space is allocated ev is the integer Event Variable
Example:	DIMENSION ICTB(7) . . . IRP=3 IUNIT=2 CALL DSKDAL (ICTB,IRP,IUNIT,IDKEV) CALL WAITFR (IDKEV)

For the system disk, the CPB for DEALLOCATE consists of the following:

Word	Contents
0	1600 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

Usually the content of first three words of the Control Table would be exactly the same as that received after some ALLOCATE request. In addition, the same values of dvcode and unit would typically be used. In general, the Control Table is established before the DSKDAL call as follows:

Word	Contents
0	Storage granted (number of words)
1	Disk platter number (bits 3-17)
2	Disk address

Bits 0-2 of word 1 need not contain the unit number. They will, in any case, be ignored and replaced within DSKDAL by the rightmost three bits of the unit parameter.

No test is made to see if the disk space to be DEALLOCATEd actually has been reserved. However, several checks are made on the Control Table arguments.

To reference any disk, the CPB for DEALLOCATE consists of the following:

Word	Contents
0	11600 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address
4	Disk type (2 for RF; 3 for RP; 24 for RK; 0 for system disk)

When a generalized DEALLOCATE is issued, the Control Table should contain the following:

Word	Contents
0	Storage granted (number of words)
1	Unit and platter numbers
2	Disk address

Word 1 of the Control Table has the following form:

Bit	Contents
0-2	Unit number
3-17	Platter number

6.7 GET: READING FROM DISK

The GET function generates a device-dependent form of the QUEUE I/O Directive which reads a specified number of contiguous words from disk into core. In addition to the Disk Driver, several I/O Handlers use GET.

The information from the Control Table is not queued along with the information from the CAL Parameter Block. Therefore, after making the I/O request, one must not modify the Control Table until after completion of the request.

As explained above, the RF, RP, or RK may serve as system disk or "user" disk. Whether a disk is dedicated to the system or the user, the MACRO and FORTRAN calls it requires for GETting words from disk do not vary. The CAL Parameter Block and Control Table, however, do vary.

Before the DSKGET call, the first three words of the table array are typically identical to the Control Table returned by a DSKAL call. The remaining four words of the Control Table are unspecified and later filled in by the DSKGET subroutine.

Following is the FORTRAN call for a disk GET:

Form:	CALL DSKGET (table,offset,words,array,dvcode, unit[,ev])
Where:	table is a seven-word integer array, the first three words of which represent a DSKAL Control Table, and the last four words of which provide space for the DSKGET Control Table offset is decimal and represents the offset from the base address of a block of disk storage at which the transfer from the disk is to begin words is decimal and represents the number of words to be transferred array is the name of the array to which data are transferred dvcode is an integer representing the code of the device from which data are obtained unit is an integer representing the unit number of the device specified by dvcode ev is the integer Event Variable
Example:	Allocate 512 words of disk storage on the disk assigned to LUN-2 and read the last 256 words into IARRAY; DIMENSION ICTB (7),IARRAY (256) . . . CALL HINF (2,IDKEV) CALL WAITFR (IDKEV) CALL UPKEV (IDKEV,IODIR,IDEV,IUNIT) . . . CALL DSKAL (ICTB,512,IDEV,IUNIT,IDKEV) CALL WAITFR (IDKEV) . . . CALL DSKGET (ICTB,256,256,IARRAY,IDEV,IUNIT, IDKEV)

If the device specified by the dvcode parameter is an RP Disk Pack or an RK Disk Cartridge, only block addressability is allowed. This requires that the offset value (a subroutine parameter) plus the disk starting address (a Control Table parameter) must equal a positive integral multiple of 256 (decimal) , i.e., the low-order eight bits of the sum must be zero. Otherwise, a negative Event Variable is returned. For the system disk, the CPB for GET consists of the following:

Word	Contents
0	3000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table associated with a GET command for the system disk is as follows:

Word	Contents
0	Disk platter number
1	Disk address
2	Core address
3	Number of words

To reference any disk, the CPB for GET consists of the following:

Word	Contents
0	13000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address
4	Disk type (2 for RF; 3 for RP; 24 for RK; 0 for system disk)

The Control Table for the generalized GET command has the following form:

Word	Contents
0	Unit and platter numbers
1	Disk address
2	Core address
3	Number of words

Word 0 of the Control Table has the following form:

Bit	Contents
0-2	Device unit number
3-17	Platter number

6.8 PUT: WRITING TO DISK

The PUT function generates a device-dependent form of the QUEUE I/O Directive which writes a specified number of contiguous words from core to disk. The function is allowed for USER- as well as EXEC-mode Tasks to perform random access. In addition to the Disk Driver, several I/O Handlers use PUT.

The information from the Control Table is not queued along with the information from the CAL Parameter Block. Therefore, after making the I/O request, one must not modify the Control Table until after completion of the request.

As explained above, the RF, RP, or RK may serve as system disk or "user" disk. Whether a disk is dedicated to the system or the user, the MACRO and FORTRAN calls it requires for PUTting data on the disk do not vary. The CAL Parameter Block and Control Table, however, do vary.

Before the DSKPUT call, the first three words of the table array are typically identical to the Control Table returned by a DSKAL call. The remaining four words are unspecified and later filled in by the DSKPUT subroutine.

Following is the FORTRAN call for a disk PUT:

Form:	CALL DSKPUT (table,offset,words,array,dvcode, unit[,ev])
Where:	table is a seven-word integer array, the first three words of which represent a DSKAL Control Table, and the last four words of which provide space for the DSKPUT Control Table offset is decimal and represents the relative position within a block of allocated disk storage at which the transfer to the disk is to begin words is decimal and represents the number of words to be transferred array is the name of the array from which data are transferred dvcode is an integer representing the code of the device from which data are obtained unit is an integer representing the unit number of the device specified by dvcode ev is the integer Event Variable
Example:	Allocate 1280 words of disk storage on the disk assigned to LUN-2 and write out 256 words onto the disk from IARRAY. Begin writing 128 words beyond the starting address of the disk storage area: DIMENSION ICTB (7),IARRAY(256) . . . CALL HINF (2,IDKEV) CALL WAITFR (IDKEV) CALL UPKEV (IDKEV,IODIR,IDEV,IUNIT) . . . CALL DSKAL (ICTB,1280,IDEV,IUNIT,IDKEV) CALL WAITFR (IDKEV) . . . CALL DSKPUT (ICTB,128,256,IARRAY,IDEV,IUNIT, IDKEV)

If the device specified by the dvcode parameter is an RP Disk Pack or an RK Disk Cartridge, only block addressability is allowed. This requires that the offset value (a subroutine parameter) plus the disk starting address (a Control Table parameter) must equal a positive integral multiple of 256 (decimal), i.e., the low-order eight bits of the sum must be zero. Otherwise, a negative Event Variable is returned.

For the system disk, the CPB for PUT consists of the following:

Word	Contents
0	3100 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table associated with a PUT command for the system disk is as follows:

Word	Contents
0	Disk platter number
1	Disk address
2	Core address
3	Number of words

To reference any disk, the CPB for PUT consists of the following:

Word	Contents
0	13100 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address
4	Disk type (2 for RF; 3 for RP; 24 for RK; 0 for system disk)

The Control Table for the generalized PUT command has the following form:

Word	Contents
0	Unit and platter numbers
1	Disk address
2	Core address
3	Number of words

Word 0 of the Control Table has the following form:

Bit	Contents
0-2	Device unit number
3-17	Platter number

6.9 UPKEV: UNPACKING AN EVENT VARIABLE

A special FORTRAN utility routine aids in constructing the device code and unit number parameters for a DSKGET or DSKPUT call. It unpacks the Event Variable returned by the HINF function. Unpacking occurs whether or not the Event Variable is negative. UPKEV is not an I/O operation and need not be followed by a WAITFOR.

It is called after a WAITFOR is done for a HINF.

The call follows:

Form:	CALL UPKEV (ev,iodir,dvcode,unit)
Where:	ev is the HINF-formatted integer Event Variable to unpack iodir is an integer and returns with bits 0-3 of ev (I/O status and directory information), right-adjusted dvcode is an integer and returns with bits 12-17 of ev (device type code), right-adjusted unit is an integer and returns with bits 4-11 of ev (unit number), right-adjusted
Example:	CALL HINF (LUN,IEV) CALL WAITFR (IEV) . . . CALL UPKEV (IEV,IODIR,IDEV,IUNIT)

CHAPTER 7

DECTAPE I/O

7.1 DT HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for DECtape operation is named DT.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is DTn for purposes of reassignment. The Handler services up to eight DECtape drives, but can handle only one file at a time.

7.2 DT HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the DT I/O Device Handler Task. Following are the legal function calls which may be issued for DT service and a brief description of each:

Table 7-1
DECtape I/O Functions

DT Call	Function
HINF	Request Handler Task information
ATTACH	Obtain exclusive use of DECtape drive
DETACH	Release attached DECtape drive
READ	Read input into buffer
WRITE	Write output from buffer
SEEK	Open file for input
ENTER	Open file for output
CLOSE	Close file
ABORT	Abort I/O for a Task
DISCONNECT & EXIT	Disconnect Handler from interrupt line and exit
GET	Read from DECtape
PUT	Write to DECtape

Except for GET and PUT, all the functions listed above are basic I/O calls and operate almost exactly as described in Chapter 2. When requested by the DT Handler Task, however, these functions do require the few special considerations discussed below.

7.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +34xx04 to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 3 to indicate input and output device
3	Set 1 to indicate a directory-oriented Handler
4-11	Set to xx, where xx is the device unit number
12-17	Device code 4 (DECtape)

7.2.2 ATTACH: Obtaining Exclusive Use of a Device

If there is an open file on the DECtape drive to be ATTACHed, the request is rejected.

7.2.3 DETACH: Releasing an Attached Device

If there is an open file on the DECtape drive to be DETACHed, the request is rejected.

7.2.4 READ: Reading Input into a Buffer

READ is legal only after a SEEK on a file. Reading an EOF effects a CLOSE. This function supports IOPS BINARY (Mode 0), IMAGE BINARY (Mode 1), IOPS ASCII (Mode 2), and IMAGE ASCII (Mode 3).

7.2.5 WRITE: Writing Output from a Buffer

WRITE is legal only after an ENTER on a file. The function supports IOPS BINARY, IMAGE BINARY, IOPS ASCII, and IMAGE ASCII.

7.2.6 SEEK: Opening a File for Input

The SEEK function searches the DECtape directory and, if the file is found, reads the first block and opens the file for input. If no ATTACH to the DECtape has occurred, the Handler issues its own ATTACH, which remains effective only while the file is open. Only one file may be open at a time.

7.2.7 ENTER: Opening a File for Output

The ENTER function searches the DECtape directory for an empty file slot and an available DECtape block. The search for a DECtape block begins at block 103. If the drive is not currently attached, the DECtape Handler issues its own ATTACH, which remains effective only while the file is open. Only one file may be open at a time.

7.2.8 CLOSE: Closing a File

The CLOSE function clears a "File Open Switch" (making SEEK and ENTER legal) and executes a DETACH if no ATTACH has been requested by the calling Task.

7.2.9 DISCONNECT & EXIT: Clearing the Interrupt Line

The Handler cannot DISCONNECT & EXIT from one unit unless no files are open on any unit. The function checks that the Reassign inhibit flag is set for all units before DISCONNECTing.

7.3 GET: READING FROM DECTAPE

The GET function causes reading from DECtape, but is only legal as long as no files are open. Only entire blocks may be read. Blocks read in the reverse direction are the complement obverse of those read in the forward direction.

GET is device-dependent. The CPB for DECtape is as follows:

Word	Contents
0	3000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Block information
4	Core buffer address
5	Word count

Word 3 consists of the following:

Bit	Contents
0	Direction to read (0 for forward, 1 for reverse)
1-17	Block number

PUT

7.4 PUT: WRITING TO DECTAPE

The PUT function causes writing to DECtape, but is only legal as long as no files are open. Only entire blocks may be written. If only part of a block is written, the state of the remainder of the block is not guaranteed. Blocks written in the forward direction should be read in that direction; blocks written in reverse should be read that way. Otherwise the data appears in complement obverse form.

PUT is device-dependent. The CPB for DECtape is as follows:

Word	Contents
0	3100 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Block information
4	Core buffer address
5	Word count

Word 3 consists of the following:

Bit	Contents
0	Direction to write (0 for forward, 1 for reverse)
1-17	Block number

CHAPTER 8

MAGTAPE I/O

8.1 MT HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for Magtape is named MT.... and handles calls submitted through special forms of the QUEUE I/O Directive. This Handler Task services DECmagtape transports connected to the TC59 Magtape Controller, both 7- and 9-track, in a manner compatible with industry standards. The device name is MTn for reassignment purposes.

The Magtape Handler services up to eight units in round-robin order. After completing a requested function, the Handler moves to the next ready unit as soon as the Magtape controller is ready. This allows overlapped operation during settling-down time.

If a unit is rewinding, it is not considered ready. The Handler does not wait on a "not ready" unit unless it is attempting error recovery. In that case it prints a message such as the following on LUN-3:

```
MT 0 NOT READY
```

The unit number is set appropriately. Error recovery is described in detail at the end of this chapter. Before performing any Magtape operations, the user Task should establish the operating mode by issuing a FORMAT request. If this is not done, the prevailing parity, density, etc., will be those which were last used on the given LUN.

8.2 MT HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the MT I/O Device Handler Task. Following are the legal function calls which may be issued for MT service and a brief description of each:

Table 8-1
Magtape I/O Functions

MT Call	Function
HINF	Request Handler Task information
ATTACH	Obtain exclusive use of Magtape drive
DETACH	Release attached Magtape drive
ABORT	Abort I/O for a Task
READ	Read input into buffer
WRITE	Write output from buffer
RDCOMP	Read and compare record
BSPREC	Backspace one or more records
BSPFIL	Backspace one or more files
REWIND	Rewind tape
WREOF	Write end-of-file mark
FSPREC	Space forward one or more records
FSPFIL	Space forward one or more files
FSPEOT	Space forward to logical end-of-tape
FORMAT	Specify tape's density and mode
MOUNT	Mount or dismount tape
LABEL	Read or write tape label
MTGET	Read from tape directly into user's buffer
MTPUT	Write to tape directly from user's buffer

HINF, ATTACH, DETACH, and ABORT are basic I/O calls and operate almost exactly as described in Chapter 2. When requested by the MT Handler Task, however, these functions do require the few special considerations discussed below.

8.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +300x05 (octal) to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 3 to indicate input and output device
3	Set 0 to indicate a non-directory-oriented Handler
4-11	Set to x, where x is the device unit number
12-17	Device code 5 (Magtape)

8.2.2 ABORT: Aborting I/O for a Task

The ABORT function is queued in the request queue of the first Magtape physical device node encountered. The Handler empties its queues of I/O requests made by the Task, but it does not stop the current I/O function in progress.

8.3 BSPFIL: BACKSPACING ONE OR MORE FILES

The BSPFIL function is used to backspace the tape a specified number of files.

BSPFIL is issued by a FORTRAN program in the following way:

Form:	CALL BSPFIL (LUN,files[,ev])
Where:	LUN is decimal and represents the Logical Unit Number files is decimal and represents the number of files to be backspaced ev is the integer Event Variable
Example:	CALL BSPFIL (25,2,IEV)

Following is the CPB for this function:

Word	Contents
0	4300 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Number of files to be backspaced

8.4 BSPREC: BACKSPACING ONE OR MORE RECORDS

The BSPREC function is used to backspace the tape a specified number of physical records.

BSPREC is issued by a FORTRAN program in the following way:

Form:	CALL BSPREC (LUN,recs[,ev])
Where:	LUN is decimal and represents the Logical Unit Number recs is decimal and represents the number of records to be backspaced ev is the integer Event Variable
Example:	CALL BSPREC (25,60,IEV)

The following is the CPB for this function:

Word	Contents
0	4200 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Number of records to be backspaced

8.5 FORMAT: SPECIFYING TAPE FORMAT

The FORMAT function is used to specify tape density, parity, 7- or 9-track mode, error-recovery mode and core-dump mode (9-track only).

FORMAT is issued by a FORTRAN program in the following way:

Form:	CALL FORMAT (LUN,type[,ev])
Where:	LUN is a decimal number representing the logical unit number type is an integer code in the range 1 to 13 (decimal) representing the tape format ev is the integer event variable
Examples:	CALL FORMAT (25,ITYPE,IEV) CALL FORMAT (25,7,IEV)

Format codes are:

Code	Meaning
1	7-track, 200 BPI
2	7-track, 556 BPI
3	7-track, 800 BPI
4	7-track, even parity
5	7-track, odd parity
6	9-track, even parity, 800 BPI
7	9-track, odd parity, 800 BPI
8	9-track, even parity, core-dump mode, 800 BPI
9	9-track, odd parity, core-dump mode, 800 BPI
10	9-track, 800 BPI, default parity
11	7-track, 800 BPI, default parity
12	Handler error recovery (default)
13	User error recovery

Because format codes are not bit designations in an 18-bit word, formats cannot be microcoded. For example, two separate FORMAT functions must be issued to specify 7-track, 200 BPI, with user error recovery. The default parity is odd, except for 7-track BCD (ASCII READ and WRITE). Formats 1 to 3 do not alter the prevailing parity. Formats 4 to 11 do not alter the prevailing density.

Following is the CPB for this function:

Word	Contents
0	5000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Format code

8.6 FSPEOT: SPACING FORWARD TO EOT

The FSPEOT function is used to space forward to the logical end-of-tape, which consists of two successive end-of-file marks. The user is solely responsible for creating a logical end-of-tape. This mark is not necessarily equivalent to the physical end-of-tape.

FSPEOT is issued by a FORTRAN program in the following way:

Form:	CALL FSPEOT (LUN[,ev])
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable
Example:	CALL FSPEOT (25,IEV)

Following is the CPB for this function:

<u>Word</u>	<u>Contents</u>
0	4700 (I/O function code)
1	Event Variable address
2	Logical Unit Number

8.7 FSPFIL: SPACING FORWARD ONE OR MORE FILES

The FSPFIL function is used to space the tape forward a specified number of files.

FSPFIL is issued by a FORTRAN program in the following way:

Form:	CALL FSPFIL (LUN,files[,ev])
Where:	LUN is decimal and represents the Logical Unit Number files is decimal and represents the number of files to space forward ev is the integer Event Variable
Example:	CALL FSPFIL (25,2,IEV)

Following is the CPB for this function:

Word	Contents
0	4600 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Number of files to space forward

8.8 FSPREC: SPACING FORWARD ONE OR MORE RECORDS

The FSPREC function is used to space the tape forward a specified number of records.

FSPREC is issued by a FORTRAN program in the following way:

Form:	CALL FSPREC (LUN,recs[,ev])
Where:	LUN is decimal and represents the Logical Unit Number recs is decimal and represents the number of records to space forward ev is the integer Event Variable
Example:	CALL FSPREC (25,50,IEV)

Following is the CPB for this function:

Word	Contents
0	4500 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Number of records to space forward

8.9 LABEL: READING OR WRITING A TAPE LABEL

The READ/WRITE LABEL function rewinds the tape to the beginning-of-tape and then causes the first record (the tape label) to be read or written. A code included in the I/O call identifies the operation. WRITE LABEL causes a previously unlabeled tape to be declared and treated as a labeled tape. If READ LABEL is requested when a tape is assumed to be unlabeled, the first record on the tape is read and no error condition is flagged.

LABEL is issued by a FORTRAN program in the following way:

Form:	CALL LABEL (LUN,code,buf,words[,ev])
Where:	LUN is decimal and represents the Logical Unit Number code is 0 to read a label, 1 to write a label buf is an integer array representing a core buffer words is decimal and represents the number of words to be read or written ev is the integer Event Variable
Example:	CALL LABEL (25,0,IBUF,50,IEV)

The CPB for this function is:

Word	Contents
0	5200 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Read (0) or write (1) code
4	Core buffer address
5	Core buffer size

8.10 MOUNT: MOUNTING OR DISMOUNTING TAPE

The MOUNT function is used to MOUNT or disMOUNT a labeled or unlabeled tape. A subfunction code included in the I/O call identifies the specific operation to be performed.

MOUNT is issued by a FORTRAN program in the following way:

Form:	CALL MOUNT (LUN,sub[,ev])
Where:	LUN is decimal and represents the Logical Unit Number sub is the integer subfunction code and may be 1, 2, or 3 (see values below) ev is the integer Event Variable
Examples:	CALL MOUNT (25,ISUBF,IEV) CALL MOUNT (25,2,IEV)

The subfunction code may have one of three values:

Code	Meaning
1	MOUNT labeled tape
2	MOUNT unlabeled tape
3	DisMOUNT tape

Following is the CPB for this function:

Word	Contents
0	5100 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Subfunction code

A labeled tape has a record which contains some sort of identifying information, written immediately after the beginning-of-tape mark. A tape is declared to be labeled when a MOUNT labeled tape function is executed and also when a WRITE LABEL function is executed.

8.10.1 MOUNT a Labeled Tape

To MOUNT a labeled tape, the requesting Task should first type out a message to the operator telling him which tape to mount. Then, when it issues the MOUNT labeled tape request, the Magtape Handler prints out the following message on LUN-3 (example is for tape 5):

```
          *** MOUNT TAPE MT5
```

Then the Handler periodically checks the status of that tape drive and considers it ready if it is on-line and at the beginning-of-tape mark.

The tape label record can be read or rewritten only by the READ/WRITE LABEL function. If the head is positioned at the beginning-of-tape and a READ, WRITE, MTGET, MTPUT, READ/COMPARE, or SPACE FORWARD request is received, the Handler automatically spaces forward past the tape label before it begins to execute the function.

Note that because of this, one cannot, for example, WRITE a record starting from the beginning-of-tape, BACKSPACE RECORD, and expect to return to the beginning-of-tape.

8.10.2 MOUNT an Unlabeled Tape

To MOUNT an unlabeled tape, the same operations should be performed as for mounting a labeled tape.

Since the tape is declared unlabeled, the special operation to space past the beginning-of-tape label of a labeled tape is not required.

8.10.3 DISMOUNT a Tape

The TC Magtape controller does not permit complete unloading of a tape under program control. The DISMOUNT tape function is equivalent to a REWIND followed by a SPACE FORWARD RECORD. No message is printed. The SPACE FORWARD will make sense in the light of the typical operating sequence described below.

Consider that tape drive 3 holds the first of a series of tapes to be processed by a certain Task on the same drive. The Task has just finished with tape 1 and needs tape 2.

The Task issues a DISMOUNT request to the Magtape Handler, types out a message to the operator, then issues a MOUNT request. The MOUNT request is not processed until the DISMOUNT is completed, so that the message

```
          *** MOUNT TAPE MT3
```

will not be printed until after the tape has spaced forward away from the beginning-of-tape. Recall that the MOUNT function periodically checks to see if the tape is at the beginning-of-tape mark. The operator can then unload tape 1 and mount tape 2.

8.11 MTGET: READING FROM TAPE INTO USER'S BUFFER

The form and meaning of GET are device-dependent. The MTGET function is used to transfer any number of words directly from tape into the user's buffer area. Unlike READ, MTGET performs no data translation (e.g., EBCDIC to ASCII), and its buffer does not have a header word pair. A WAITFR call is required to determine completion of the operation. Several MTGETs can be active simultaneously on different LUNs. MTGET permits overlapped I/O.

Odd parity, 800 BPI, and 7-track operation are default assumptions. If the prevailing mode is 9-track, the data are read from the tape in 9-track core-dump mode.

MTGET is issued by a FORTRAN program in the following way:

Form:	CALL MTGET (LUN,buf,words,ctb[,ev])
Where:	LUN is decimal and represents the Logical Unit Number buf is the integer array into which data are transferred words is the number of integer words to be transferred ctb is a three-word integer array representing the Control Table ev is the integer Event Variable
Example:	DIMENSION ICTB(3) . . . CALL MTGET (LUN,IA,ICNT,ICTB,IEV)

Following is the MTGET CPB:

Word	Contents
0	3000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table (CTB) is as follows:

Word	Contents
0	Core buffer address
1	Number of words to be transferred; when done, the Handler replaces this with the actual number of words transferred
2	Magtape status is stored here at completion of the operation

The information from the Control Table is not queued along with the information from the CAL Parameter Block. Therefore, after making the I/O request, one must not modify the Control Table until after completion of the request.

8.12 MTPUT: WRITING TO TAPE FROM USER'S BUFFER

The MTPUT function is used to transfer words directly from the user's buffer area to tape. Unlike WRITE, MTPUT performs no data translation (e.g., ASCII to BCD), and its buffer does not have a header word pair. A WAITFR call is required to determine completion of the operation. Several MTPUTs can be active simultaneously on different LUNs. MTPUT permits overlapped I/O.

Odd parity, 800 BPI, and 7-track operation are default assumptions. If the prevailing mode is 9-track, the data are written in 9-track core-dump mode.

MTPUT is issued by a FORTRAN program in the following way:

Form:	CALL MTPUT (LUN,buf,words,ctb[,ev])
Where:	LUN is decimal and represents the Logical Unit Number buf is the integer array from which data are transferred words is the number of integer words to be transferred ctb is a three-word integer array representing the Control Table ev is the integer Event Variable
Example:	DIMENSION ICTB(3) . . . CALL MTPUT (LUN,IA,ICNT,ICTB,IEV)

Following is the MTPUT CPB:

Word	Contents
0	3100 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table (CTB) is as follows:

Word	Contents
0	Core buffer address
1	Number of words to be transferred
2	Magtape status is stored here at completion of the operation

8.13 ERROR RECOVERY

The user has the choice of two modes of operation when hardware errors occur.

8.13.1 Handler Recovery

The normal mode of operation assumed by the Magtape Handler is that it is responsible for error recovery. When an error occurs, the Handler attempts to reexecute the function in case the error is only transient. After eight more tries, it treats the error as unrecoverable.

Unrecoverable errors include parity error, bad tape, write lock (on output functions), and error caused by a tape unit that is not ready during function execution. The last condition should not occur unless the hardware fails or the operator mistakenly sets the transport not-ready.

Unrecoverable errors usually set the requester's Event Variable to -12. The one exception occurs when a parity error persists during a READ. In such a case, the data are passed on and the error is flagged in the data validity bits in the line buffer header.

8.13.2 User Recovery

When user recovery is specified, the Handler does not try to reexecute a function that has failed. Instead, the user's Event Variable is set with the Magtape status after all such fatal errors (bad tape, data late, and write lock (on output functions)). The tape is left after the last record that was read, written, or spaced over when the error occurred. If, however, the tape unit becomes not ready during function execution, the Event Variable is set to -12.

Table 8-2
ASCII to BCD Conversion

Character	ASCII	BCD	Character	ASCII	BCD
Null	0	0	-	255	40
1	261	1	J	312	41
2	262	2	K	313	42
3	263	3	L	314	43
4	264	4	M	315	44
5	265	5	N	316	45
6	266	6	O	317	46
7	267	7	P	320	47
8	270	10	Q	321	50
9	271	11	R	322	51
0	260	12	!	241	52
=,#	275,243	13	$	244	53
',"	247,242	14	*	252	54
:	272	15	]	333	55
>	276	16	;	273	56
UNDERSCORE	137	17	FORM FEED	214	57
SPACE	240	20	+,&	253,246	60
/	257	21	A	301	61
S	323	22	B	302	62
T	324	23	C	303	63
U	325	24	D	304	64
V	326	25	E	305	65
W	327	26	F	306	66
X	330	27	G	307	67
Y	331	30	H	310	70
Z	332	31	I	311	71
CAR. RET.	215	32	?	277	72
,	254	33	.	256	73
(,%	250,245	34	)	251	74
TAB	211	35	[	335	75
RUBOUT	377	36	<	274	76
LINE FEED	212	37	ALTMODE	375	77

NOTE

Where multiple characters are shown, the first is the character generated on conversion from BCD to ASCII. All characters not shown are discarded.

Table 8-3
ASCII to EBCDIC Conversion

Character	ASCII	EBCDIC	Character	ASCII	EBCDIC
Null	0	0	<	274	114
LINE FEED	212	45	=	275	176
CAR. RET.	215	25	>	276	156
SPACE	240	100	?	277	157
!	241	132	@	300	174
"	242	177	A	301	301
#	243	173	B	302	302
TAB	211	5	C	303	303
$	244	133	D	304	304
%	245	154	E	305	305
&	246	120	F	306	306
'	247	175	G	307	307
(,[	250,333	115	H	310	310
),]	251,335	135	I	311	311
*	252	134	J	312	321
+	253	116	K	313	322
,	254	153	L	314	323
-	255	140	M	315	324
.	256	113	N	316	325
/	257	141	O	317	326
0	260	360	P	320	327
1	261	361	Q	321	330
2	262	362	R	322	331
3	263	363	S	323	342
4	264	364	T	324	343
5	265	365	U	325	344
6	266	366	V	326	345
7	267	367	W	327	346
8	270	370	X	330	347
9	271	371	Y	331	350
:	272	172	Z	332	351
;	273	136	RUBOUT	377	7
UNDERSCORE	137	137			

NOTE

Where multiple characters are shown, the first is the character generated on conversion from EBCDIC to ASCII. All characters not shown are discarded.

8.14 RDCOMP: READING AND COMPARING A RECORD

The RDCOMP I/O function is used to read and compare a record after data have been transferred by a GET or PUT call or a binary READ or WRITE. The user should previously have backspaced to position the tape head before the record to be read and compared. A WAITFR call is required to determine completion of the operation. Default parameters are 7-track, 800 BPI, and odd parity. It makes little sense to READ/COMPARE without first performing one of the operations previously mentioned. RDCOMP permits overlapped I/O.

The tape record is compared directly with the contents of the user's core buffer. Because the tape contains BCD or EBCDIC characters, tape records cannot be compared to ASCII-mode core buffer data. Several RDCOMPs can be active simultaneously on different LUNs.

RDCOMP is issued by a FORTRAN program in the following way:

Form:	CALL RDCOMP(LUN,buf,words,ctb[,ev])
Where:	LUN is decimal and represents the Logical Unit Number buf is an integer array to which data are transferred words is decimal and represents the number of words to be transferred ctb is a three-word integer array representing the Control Table ev is the integer Event Variable
Example:	DIMENSION ICTB(3) • • • CALL RDCOMP(25,IBUFF,50,ICTB,IEV)

Following is the CPB for this function:

Word	Contents
0	4000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table is constructed as follows:

Word	Contents
0	Core buffer address
1	Decimal number of words to compare
2	Magtape status

Magtape status (word 2) is returned on completion of the operation.

8.15 READ: READING INPUT INTO A BUFFER

The READ I/O function READs input into a buffer via the MT Handler Task assigned to the specified Logical Unit Number. It operates almost exactly as in the description in Chapter 2, with the exceptions and special features described below.

Both 7- and 9-track binary tapes as well as 9-track ASCII tapes are read in odd parity. Seven-track ASCII is read in even parity. Nine-track binary is written in core dump mode, a recording method employed on 9-track drives. The parity and density may be overridden by a FORMAT request. The default is 800 BPI (bits per inch).

If an end-of-file mark is encountered, this condition is flagged in the standard manner in the line buffer header word 0. In addition, the requester's Event Variable is set to +3.

If end-of-tape is passed during a READ, the function is performed and the Event Variable is set to +4. If both end-of-tape and end-of-file occur, the end-of-tape indication takes precedence.

On completion of the READ request, the data validity bits in line buffer header word 0 are set nonzero if one of the following errors has occurred. Because only one condition can be flagged, they are listed in order of precedence:

1. Parity error (checksum not computed)
2. Short line buffer error (checksum not computed)
3. Checksum error

8.15.1 IOPS BINARY (Mode 0)

The format of data returned to the requester is standard IOPS BINARY (including the header word pair). Data are transferred from the tape directly to the user's core buffer.

8.15.2 IMAGE BINARY (Mode 1)

Use of the IMAGE BINARY data format means that only binary data are read from the tape (directly into the requester's buffer). The header word pair, which is not written on the tape in this mode, is constructed by the Handler and stored in the user's buffer header.

8.15.3 IOPS ASCII (Mode 2)

Because ASCII characters are not written on the tape, the Handler must convert from one character set to another when data are read in. The record is read in even parity BCD for a 7-track tape and odd parity EBCDIC for 9-track tape. These are converted to ASCII characters and packed into the requester's line buffer in the standard IOPS ASCII form. If there are not enough tape characters to fill the last word pair completely, the last character positions are set to zero (null characters). The I/O Handler computes and sets the header word pair in the line buffer.

8.15.4 IMAGE ASCII (Mode 3)

READ in this mode unpacks a tape record in BCD or EBCDIC in the same manner as for IOPS ASCII, except for the way in which the characters are stored in the user's line buffer. For IOPS ASCII, characters are packed five for every two words of memory. In IMAGE ASCII, which is sometimes a programming convenience, only one character is stored per word. Records written in IOPS ASCII can be read in IMAGE ASCII. The Handler computes and sets the line buffer header word pair.

8.16 REWIND: REWINDING THE TAPE

The REWIND function is used to REWIND the tape and position the tape head at the beginning-of-tape mark.

If the user wants to REWIND from a FORTRAN task, no subroutine call is required. He can simply use the standard FORTRAN REWIND statement.

Following is the CPB for this function:

Word	Contents
0	4100 (I/O function code)
1	Event Variable address
2	Logical Unit Number

8.17 WREOF: WRITING AN END-OF-FILE MARK

The WREOF function is used to write an end-of-file mark on the tape at the parity and density of the file preceding the mark. In particular, writing in 9-track binary requires an explicit FORMAT request so that the end-of-file mark is written in core-dump mode.

If the user wants to write an end-of-file mark from a FORTRAN Task, no subroutine call is required. He can simply use the standard PDP-15 FORTRAN ENDFILE statement.

Following is the CPB for this function:

Word	Contents
0	4400 (I/O function code)
1	Event Variable address
2	Logical Unit Number

8.18 WRITE: WRITING OUTPUT FROM A BUFFER

The WRITE function call WRITEs output from a buffer to the MT Handler Task assigned to the specified Logical Unit Number. It operates almost exactly as described in Chapter 2, with the exceptions and special features discussed below.

Both 7- and 9-track binary tapes as well as 9-track ASCII tapes are written in odd parity; 9-track binary is written in core dump mode. A FORMAT request is mandatory prior to writing 9-track binary; otherwise the end-of-file mark may be unreadable. 7-track ASCII is written in even parity. The parity and density may be overridden by a FORMAT request. The default density is 800 BPI.

If end-of-tape is passed during a WRITE, the function is performed and the requester's Event Variable (if indicated) is set to +4. After that, WRITE requests are illegal.

8.18.1 IOPS BINARY (Mode 0)

The IOPS BINARY record is written directly to the tape from the user's line buffer, including his line buffer header word pair. First, however, the Magtape Handler computes the checksum and stores it in line buffer header word 1. Since the data are written directly from the user's core area, he must not modify the contents of the line buffer until the transfer is completed. The tape is written in odd parity and an even number of words is transferred.

8.18.2 IMAGE BINARY (Mode 1)

The IMAGE BINARY record is transferred to the tape directly from the user's line buffer without the line buffer header. Consequently, the buffer should not be altered until after the transfer is completed. The tape is written in odd parity and an even number of words is transferred.

8.18.3 IOPS ASCII (Mode 2)

The IOPS ASCII data is moved from the user's line buffer into a buffer internal to the Magtape Handler. There it is unpacked and converted into industry standard BCD (7-track) or EBCDIC (9-track). Null (zero) characters following the last ASCII character in the line buffer are not suppressed and no header word pair is recorded. The maximum record size is 376 octal (254 decimal). Characters are written one per tape frame. The BCD and EBCDIC characters are industry-compatible where possible (see conversion charts). The 7-track tapes may be read on industry-compatible machines with Data Translator off and with Data Converter off.

8.18.4 IMAGE ASCII (Mode 3)

IMAGE ASCII data are packed one per word in the user's line buffer. Otherwise, they are treated identically to IOPS ASCII when written to tape.

CHAPTER 9

TERMINAL I/O

9.1 TTY HANDLER TASK

The multiterminal I/O Device Handler Task responsible for servicing requests for terminals is named TTY and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is TTn for purposes of reassignment.

This Handler Task controls I/O to the console terminal TTY and to the devices connected to LT15 or LT19D control hardware. Note that the LT15 Teletype control is indistinguishable from the LT19D from a software point of view.

A maximum of seventeen terminals, including the console terminal, can be supported. This is not an inherent software limitation. Therefore, if a replacement for the LT19D is built that can support, for example, 23 terminals, it would be easy to convert the Terminal Handler accordingly. The limitation would be the total effective baud rate.

A variety of terminals are supported by XVM/RSX. All terminals in the system must be defined by the DTC (define Terminal Characteristics) MCR function.

9.2 TERMINAL CHARACTERISTICS

A terminal may be characterized as send-only (to the CPU), receive-only (from the CPU), or send-receive. Terminals that can both send and receive operate in full duplex mode so that input does not mix with output. At present, it is assumed that the terminal does not have a local copy mechanism. This means that the software (here the TTY Task) must send characters input by the terminal back to its "printer." The console terminal operates in full duplex mode and local copy is suppressed by the use of the KRS instruction rather than KRB. The local copy facility should not be confused with the term "half duplex."

RSX cannot necessarily support any device that can be physically connected to the LT19D.

Most keyboard/printer devices have special form control functions which cannot be performed within a single character time, e.g., horizontal tab, vertical tab, form feed, and bell. Usually the hardware does not provide the necessary delay to ensure that such functions have completed before printing characters are transmitted. For these devices, the TTY task provides the delay by following the control character with a specific number of fillers (usually the null character). TTY generates fillers for horizontal tab, vertical tab, bell and form feed for both input and output in both IMAGE ASCII and IOPS ASCII data modes. Filler characters are generated as needed and are not stored in the requester's buffer.

Horizontal tab, vertical tab and form feed mechanisms are assumed to exist on all output terminals, but RSX does not depend on this. This means that these functions are not simulated by the software in the absence of the hardware. However, for editing and listing assembly language programs, horizontal tab is a necessity. For proper operation, horizontal tab stops should be eight spaces apart, vertical tab stops should be six lines apart and the form-feed stop should coincide with a vertical tab stop. A form (or page) should contain 66 lines.

All output terminals should have an audible alarm, such as a bell, for ASCII code 07 (CTRL/G). This facility is not now used by RSX, but it might be used in the future.

The paper tape input facility on KSR terminals is not supported, chiefly because the software cannot read one character at a time on demand. Therefore, it is possible for characters to pass through the reader unnoticed. ASCII tapes can be punched on KSR terminals and can be read by the PC15 high-speed paper tape reader handler. Oil-base tape, however, cannot be read photoelectrically, because it transmits too much light. The reader handler recognizes seven-bit code 04 (CTRL/D or EOT) as the end-of-file when reading in IOPS ASCII. Code 04 is punched by the PC15 punch handler when closing an IOPS ASCII file, but the TTY handler ignores the CLOSE function. Hence, it is useful to have an end-of-file tape handy (a tape consisting of a single eight-bit character: 204).

Characters are normally output as seven-bit quantities without parity. If output parity is required, the RSX Executive should be reassembled with the PARITY parameter defined (set equal to any value).

9.3 LEGAL DATA MODES

IOPS ASCII (mode 2) and IMAGE ASCII (mode 3) are legal for both READ and WRITE functions.

IMAGE ASCII, as treated by the terminal handler, is not a pure image mode. That is, the user program does not have complete control over what is sent to or received from a device. The handler supplies timing (filler) characters following horizontal tab, vertical tab, form feed and bell, even in IMAGE ASCII mode.

Also, on output, null, rubout and altmode characters are usually screened and not printed. Nulls and rubouts are not printed, so they can be used as the last character(s) in an IMAGE ASCII output line buffer. In particular, this means that output of one character at a time can occur at full speed (recall that the word-pair count is used, which implies an even number of characters in the buffer).

9.4 TYPE-IN

For each terminal, only one input character can be processed at a time (i.e., there is no type-ahead buffering scheme). This means that characters typed might be ignored. With the exception of special control characters (CTRL/C, CTRL/P, CTRL/Y, CTRL/T, CTRL/X and CTRL/U), this occurs when the terminal has not received a READ command or when the character previously typed has not been processed. The latter situation can occur when a form feed is followed by printing characters.

Because send/receive terminals are operated in full-duplex mode, the typist can tell that a character has been ignored when it is not immediately displayed.

Input characters are read as eight-bit values and are truncated to seven bits.

9.4.1 CTRL/C from the MCR Terminal

One of the terminals that has a keyboard is designated as the MCR device. The Monitor Console Routine can be invoked (by typing CTRL/C) from this unit only.

Whenever CTRL/C is typed at the MCR device, the interrupt service routine in the terminal handler (TTY) examines a software flag called MCRRI (the MCR request inhibit flag). When MCRRI is zero, neither the Resident MCR task (...MCR) nor any of the disk-resident MCR function tasks is active. In this case, TTY sets MCRRI to one and requests the Resident MCR task.

If CTRL/C is typed and MCRRI is nonzero, indicating that an MCR function is active, the Resident MCR task is not requested and MCRRI is set to -1. Several MCR function tasks (those that are capable of producing lengthy output) use the fact that MCRRI is -1 to prematurely terminate output. Therefore, CTRL/C has a dual function in the system.

The operator can redefine the MCR device by using the REASSIGN MCR task as follows:

```
          MCR>REA 2,3 TT1 TT0
```

9.4.2 CTRL/T to Invoke MULTIACCESS

Whenever CTRL/T is typed at terminal "nn", TTY requests the MULTIACCESS Monitor task, TDV..., and sets bit nn of memory location 226 (MA.CT). The change of state in SCOM word MA.CT indicates to TDV... that the user at terminal nn requires service.

9.4.3 CTRL/X for Remote Stimulus

● Whenever CTRL/X is typed at terminal "nn", TTY requests a task called TTY.nn. For example, if CTRL/X is typed on TT3, a task named TTY.03 is requested. If the task does not exist or if it is already active, CTRL/X is ignored.

This feature is independent of any TTY READ or WRITE requests that might be pending or in progress. It provides the user with a means of signaling to the system that he is at a specific terminal and that he wants attention.

9.4.4 CTRL/Y to Abort a User Task Under MULTIACCESS

Whenever CTRL/Y is typed at terminal "nn", TTY sets bit nn of memory location 227 (MA.CY). The change of state in SCOM word MA.CY indicates to TDV... that the user at terminal nn wishes to abort his current task.

9.4.5 CTRL/P to Resume the Task USR.nn

Whenever CTRL/P is typed at terminal "nn", TTY issues a RESUME directive for the task named USR.nn. This facility enables users to resume tasks suspended under MULTIACCESS.

9.5 TTY HANDLER REQUESTS

Special forms of the QUEUE I/O directive can be issued to queue requests to TTY. Following are the legal function calls that can be issued for terminal service, with a brief description of each:

Table 9-1
Terminal I/O Functions

TTY Call	Function
HINF	Request handler task information
ATTACH	Obtain exclusive use of terminal
DETACH	Release attached terminal
READ	Read input into buffer
WRITE	Write output from buffer
ABORT	Abort I/O for a task

HINF, ATTACH and DETACH are basic I/O calls and operate almost exactly as described in Chapter 2. However, ATTACH and DETACH are essentially ignored for user-mode tasks with mapped LUNs (i.e., all user-mode tasks run under MULTIACCESS), except that the specified event variable, if any, is set to +1.

When requested by the terminal handler task, HINF requires the special considerations described in the following paragraph.

9.5.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the event variable is set to +y0xx01 to indicate the following:

Field	Bit	Contents
+y	0	Set to 0 to make the event variable positive
	1-2	Set to y, where y is 1 if the device allows only output, 2 if only input, 3 if both
0xx	3	Set to 0 to indicate a non-directory-oriented handler
	4-11	Set to xx, where xx is the device unit number
01	12-17	Set to device code 01 (terminal)

9.6 ABORT: ABORTING I/O FOR A TASK

Unlike handlers for other low-speed devices, TTY does not wait for completion of any I/O. Consequently, the ABORT request is dequeued in the normal way. ABORT is queued in only the first terminal node in the Physical Device List. The handler, however, must abort I/O by the indicated task on each terminal unit, provided that the ABORT request was queued by the I/O Rundown task, IORD.

For each terminal, the following operations are performed:

1. If the task has attached the device, a DETACH operation is performed.
2. Any task-made I/O requests that remain in the I/O request queue for this terminal unit are removed and the nodes are returned to the pool of empty nodes.
3. If I/O by the task is in progress to this terminal unit, it is terminated.

Unlike other I/O handlers, TTY permits tasks to abort their own I/O for a specified LUN. Such ABORT requests are handled in the same way as requests by IORD, except that ABORT requests by IORD abort all I/O operations for the specified task on all terminal units. ABORT requests from other tasks abort I/O operations for only the LUN specified.

9.7 READ: READING INPUT INTO A BUFFER

The READ function call reads input into a buffer via the terminal handler task assigned to the specified logical unit number. It is legal only if the terminal can send data to the CPU. It operates almost exactly as described for READ in Chapter 2, with the exceptions and special features described below.

If the last READ/WRITE command was in IOPS ASCII and was terminated by a carriage return, a line feed is output to move off the last printed line.

9.7.1 IOPS ASCII (Mode 2)

This section summarizes the characteristics of IOPS ASCII mode relevant to terminal input:

1. Characters are packed in the requester line buffer in 5/7 ASCII format, and the line buffer header is set to reflect the data mode (2) and the number of word pairs actually read in.

2. Codes 03 (CTRL/C), 24 (CTRL/T), 25 (CTRL/U), 30 (CTRL/X), 20 (CTRL/P), 31 (CTRL/Y), 177 (rubout), 33 and 176 cannot be read into the requester line buffer. Codes 33 and 176 are alternate forms of altmode. They are converted to 175, the standard internal representation of altmode. Code 176 may become available, because it is used by older terminals.

3. The line buffer size given in the I/O call must be at least +4. An odd size is truncated to an even size (without changing the requester CAL parameter block). In other words, the odd word is ignored. The largest word-pair count that can be recorded in a line buffer header is 377. If the buffer size is larger than 776 (twice 377), the size is considered to be 776. For a user-mode task, a check is made to ensure that the buffer lies entirely within the task partition.

4. The editing functions CTRL/U and rubout apply. CTRL/U is printed as "@" and signifies that the line buffer has been reset to empty. Rubout is ignored if the line buffer is empty. If not, it is printed as "\" and it erases the last character in the line buffer.

5. Normal line termination occurs when an altmode or carriage-return character is received. Because it is often desirable to know which terminator it is without having to scan the line, the requester event variable is set to +1 for altmode and +2 for carriage return

6. Abnormal line termination occurs when too many characters are received, causing buffer overflow. Buffer overflow occurs when the buffer is full except for one character, and the next character received is a character other than rubout, CTRL/U, carriage return or altmode. This causes the final character to be replaced by a carriage return (which is printed) and causes the line buffer header validity bits (12 and 13) to be set to 11.

7. CTRL/D is not a special character. The standard method of inputting end-of-file from a terminal in RSX is to type a line consisting of two characters: either CTRL/D and carriage return or CTRL/D and altmode.

9.7.2 IMAGE ASCII (Mode 3)

This section summarizes characteristics of IMAGE ASCII mode relevant to terminal input:

1. Seven-bit characters are stored one per word, right-justified in the requester line buffer. The line buffer header is set to reflect the data mode (3) and the number of word pairs actually read in. If an odd number of characters is input, the second word of the last word pair used is not modified by TTY.

2. Codes 03 (CTRL/C), 24 (CTRL/T), 30 (CTRL/X), 33, 20 (CTRL/P), 31 (CTRL/Y) and 176 cannot be read into the requester line buffer. Codes 33 and 176 are alternate forms of altmode. They are converted to 175, the standard internal representation of altmode. Code 176 may become available, because it is used by older terminals. Codes 25 (CTRL/U) and 177 (rubout), which are not passed on to the requester if the data mode is IOPS ASCII, are stored in the line buffer if the mode is IMAGE ASCII.

3. The line buffer size given in the I/O call must be at least +3. The largest word-pair count that can be recorded in a line buffer header is 377. If the buffer size is larger than 776 (twice 377), the size is considered to be 776. For a user-mode task, a check is made to ensure that the buffer lies entirely within the task partition.

4. The editing functions CTRL/U and rubout, which are used in IOPS ASCII, do not apply in IMAGE ASCII. Those characters are simply recorded in the requester line buffer.

5. CTRL/D is not a special character. The standard method for inputting end-of-file from a terminal in RSX is to type a line consisting of two characters: either CTRL/D and carriage return or CTRL/D and altmode.

9.8 WRITE: WRITING OUTPUT FROM A BUFFER

The WRITE I/O function call WRITEs output from a buffer to the Terminal Handler Task assigned to the specified Logical Unit Number. It is legal only if the terminal can receive data from the CPU. It operates almost exactly as in the description of WRITE in Chapter 2, with the exceptions and special features described below.

9.8.1 IOPS ASCII (Mode 2)

This section summarizes characteristics of IOPS ASCII mode relevant to terminal output:

1. Seven-bit characters are unpacked from the requester's line buffer in 5/7 ASCII (IOPS ASCII) format. TTY outputs an 8-bit, even-parity character, which certain devices may require.

2. Codes 00 (null), 175 (ALTMODE), and 177 (rubout) are not printed if they come from the requester's line buffer.

3. Line termination is based on the occurrence of a carriage return or ALTMODE character.

4. The editing function CTRL/U applies to IOPS ASCII output. If CTRL/U is typed in while the terminal is performing IOPS ASCII output, the remainder of the line is not printed, and carriage return is printed in place of CTRL/U.

   Normally, line termination causes the requester's Event Variable to be set to +1; however, it is set to +2 when CTRL/U aborts the output.

5. IOPS ASCII lines constructed by the FORTRAN Object-Time System (OTS) begin with a vertical form control character and terminate with a carriage return, e.g.,

   ```
   <12> TEXT <15>
   ```

   The vertical form control characters are: line feed (12), vertical tab (13), form feed (page eject) (14), overprint (20), and double space (21). OTS does not use vertical tab. Overprint and double space are not the meanings assigned to codes 20 and 21 in the USASCII Standard.

   Traditionally, normal output lines constructed by MACRO language programs do not contain an initial line feed, e.g.,

   ```
   TEXT <15>
   ```

   Therefore, carriage return has always meant carriage return-line feed in IOPS ASCII mode.

   However, in order to implement overprint, the implied line feed cannot be output until the next READ/WRITE command is processed. For IOPS ASCII WRITE, the first character in the line buffer is examined. If it is not a line feed or overprint, and if the last READ/WRITE command was also in IOPS ASCII mode and terminated with a carriage return, a leading line feed is printed by TTY. On the other hand, if the last READ/WRITE command was in IMAGE ASCII mode, it is assumed that the line terminated with carriage return-line feed. Therefore, if the first IOPS ASCII character is a line feed, it is discarded to prevent unwanted double spacing.

Code 21, alias double space, is handled only to satisfy the output format requirements of FORTRAN IV. Even though it is a leading form control character, TTY first prints a line feed. Then, code 21 is replaced by code 12 (line feed) so that a second line feed is printed. Double spacing occurs only when code 21 appears at the beginning of an IOPS ASCII output line, assuming that the previous line was also in IOPS ASCII and ended with a carriage return. All other occurrences of the double space character cause only a single line feed.

9.8.2 IMAGE ASCII (Mode 3)

This section summarizes characteristics of IMAGE ASCII mode relevant to terminal output.

1. Seven-bit characters are taken right-justified, one per word, from the requester's line buffer. TTY converts them to 8-bit, even-parity characters, which certain devices may require.

2. If the last READ/WRITE command was in IOPS ASCII and was terminated by a carriage return, a line feed is output to move off the last printed line.

3. Codes 00 (null), 175 (ALTMODE), and 177 (rubout) are not printed if they come from the requester's line buffer.

4. The word-pair-count is taken from the line buffer header and must be at least +2. If the Task runs in USER-mode, a check is made to ensure that the buffer lies entirely within the Task's partition. To output an odd number of characters, the last character in the buffer should be a null (zero), which is not printed.

5. The editing function CTRL/U, which applies to IOPS ASCII output, does not apply to IMAGE ASCII output.

#175 WILL PRINT IF TERMINAL TYPE DEFINED AS [illegible]

CHAPTER 10

CARD READER I/O

10.1 CD HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for the card reader is named CD.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device code is CD for reassignment purposes.

The CD Handler Task supports all card readers connected to the CR15 Controller and through the UC15 Unichannel as well as the CR03B. The Handler's default is 029 punched card formats and must be reassembled for 026 format (see the procurement document).

10.2 CD HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the CD I/O Device Handler Task. Following are the legal function calls which may be issued for CD service and a brief description of each:

Table 10-1
Card Reader I/O Functions

CD Call	Function
HINF	Request Handler Task information
ATTACH	Obtain exclusive use of card reader
DETACH	Release attached card reader
ABORT	Abort I/O for a Task
DISCONNECT & EXIT	Disconnect Handler from interrupt line and exit
READ	Read input into buffer

HINF, ATTACH, DETACH, ABORT, and DISCONNECT & EXIT are basic I/O calls and operate almost exactly as described in Chapter 2. When requested by the CD Handler Task, however, HINF does require the few special considerations discussed below.

10.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +200007 (octal) to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 2 to indicate input-only device
3	Set 0 to indicate a non-directory-oriented Handler
4-11	Device unit 0
12-17	Device code 7 (card reader)

10.3 READ: READING INPUT INTO A BUFFER

The READ I/O function READs input into a buffer via the CD Handler Task assigned to the specified Logical Unit Number. It operates almost exactly as in the description of READ in Chapter 2, with the exceptions and special features described below.

A card image is read, a carriage return character is appended, and the modified image is packed into a requester-specified IOPS ASCII (data mode 2) line buffer. The following CPB is used to queue a READ request:

Word	Contents
0	2600 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	I/O data mode (2)
4	Line buffer address
5	Buffer size (in words)

Eighty card columns are read and interpreted as Hollerith data (029 or 026 code, depending upon the version of the Handler), mapped into the corresponding 65-graphic subset of ASCII (DEC029 or DEC026 code) and stored in the user's line buffer in 5/7 format. Compression of internal blanks (spaces) to TABs and truncation of trailing blanks are not performed. A carriage return character (015 octal) is appended to the input line; thus a total of 81 characters is returned by the Handler in IOPS ASCII mode.

The single addition to the Hollerith set, necessitated by the constraints of system programs, is the provision for the internal generation of the ALTMODE terminator. The appearance of a 12-1-8 punch on the card is mapped into the standard ALTMODE character (175 octal) in the user's line.

Another special punch configuration is the end-of-file card (all rows punched in column 1). When an end-of-file card is encountered, the remainder of the card is ignored and the Handler returns the standard EOF designation (1005 in word 0 of the line buffer header) to the user's program.

10.4 CR15 ERROR MESSAGES

Corrective action can be taken for a variety of malfunctions. These are therefore treated as a form of non-terminal error with recovery as described below. Error messages ordinarily appear on LUN-64, but the user can conditionally assemble the Handler so that they appear on a different LUN.

1. *** CD READER NOT READY

   Cause: Hopper empty, READ "STOP" button pushed, or power off.

   Corrective action: Place card in input hopper, push "POWER" on button and/or push "RESET" button. (The reader takes a few seconds to build up its air foil.)

2. *** CD PICK ERROR

   Cause: Card failed to leave input hopper on READ request.

   Corrective action: Juggle cards in input hopper and depress "RESET" button.

3. *** CD DATA MISSED/PHOTO ERROR

   Cause: Loss of power while reading a card or hardware malfunction in photo sensing devices.

   Corrective action: Remove top card from output stacker, place card at bottom of card deck in input hopper. Depress "POWER" button and/or depress "RESET" button.

4. *** CD ILLEGAL PUNCH

   Cause: Illegal card punch detected on last card read. (Hollerith code not part of DEC029 or DEC026 set.)

   Corrective action: Depress "STOP" button to disable card reader. Remove top card from output stacker, punch new card to correct columns containing incorrect Hollerith code, place corrected card at bottom of card deck in input hopper and, finally, depress "RESET" button to enable card reader.

NOTE

When transferring a card deck to any device, be sure to include an end-of-file card at the end of the input card deck.

10.5 UC15 OPERATION

To a very large extent, the UC15 readers are programmed and operated exactly as the CR15 readers are. The primary difference is the presence of spooling in the PDP-11. When this feature is enabled (with the DOS SPOOL function), card images proceed from the card reader to the user's program via the RK05 disk. This means that a large deck of cards can be fed into the card reader and onto the RK05 prior to their use in the XVM. Indeed, CD.... need not event be core-resident at the time of the initial card reading. When the cards are needed, they can be read from the RK05 at disk speed. This whole operation is transparent to the user program.

A special card is needed as a deck delimiter for the Spooler. This is called an end-of-deck card. This EOD card is used in addition to the ordinary EOF card. It has ALTMODE punches (12-1-8) in columns 1 and 2. In general, each job nas several decks, each ending with an EOF card; an EOD card follows the entire job.

If desired, spooling can be disabled by assembling CD.... with the assemoly parameter NOSPL=0. To obtain a UC15 Handler, independent of spooling, the assembly parameter UC15=0 is needed.

10.6 UC15 ERRORS

The basic mechanism of error handling and recovery for the UC15 Unichannel readers is similar to that of the CR15: the user corrects the error, and the Handler resumes operation when the reader becomes ready. The types of errors and recovery procedure are also similar. The error printout format, however, is different.

The PDP-11 handles errors. Any errors that occur are placed in a table in the PDP-11. An RSX monitor program in the XVM called the POLLER transfers any such entries to the MCR terminal. The message format is:

```
        *** UC15 ERROR CDU xxxyyy
```

where xxx is the Spooler error code, and yyy is the card reader error code. A Spooler error code of 004 means that the card Spooler was empty when it received a READ. The card reader error codes are:

Code	Meaning
074	Column done interrupt before previous one serviced
012	Read check (covers a variety of hardware problems)
072	Illegal punch combination
004	Reader off line (not returned if spooling): a number of causes, usually a hopper or stacking error.
075	Hardware is busy, but driver is not
076	Hardware error between cards
045	More than 80 columns found on card
003	Illegal interrupt, or unexpected reader-to-on-line

Table 10-2
DEC029 Character Set
Hollerith Card Code - Type 29 Punch

Digit	Zone			
	None	12	11	0
None	Space	&	-	0
1	1	a	j	/
2	2	b	k	s
3	3	c	l	t
4	4	d	m	u
5	5	e	n	v
6	6	f	o	w
7	7	g	p	x
8	8	h	q	y
9	9	i	r	z
8-2	:	([)	!	(])
8-3	#	.	$	,
8-4	@	<	*	%
8-5	'	(	)	(←)
8-6	=	+	;	>
8-7	"	(↑)	(\)	?

Special characters:

ALTMODE = 12-1-8 multiple punched

end-of-file = All rows punched in column 1

NOTE

In the table above, the following punch configurations have graphic representations which at present do not conform to DEC standards. The characters shown in parentheses (above) indicate the ASCII equivalents in the DEC029 Character Set:

12-8-2	cent sign
0-8-5	underscore
12-8-7	vertical bar
11-8-7	logical NOT sign

Table 10-3
DEC026 Character Set
Hollerith Card Code - Type 26 Punch

Digit	Zone			
	None	12	11	0
None	Space	+	-	0
1	1	a	j	/
2	2	b	k	s
3	3	c	l	t
4	4	d	m	u
5	5	e	n	v
6	6	f	o	w
7	7	g	p	x
8	8	h	q	y
9	9	i	r	z
8-3	=	.	$	,
8-4	@	)	*	(
Card Punch	Combinations Below Multiple Punched			
8-2	(←)	(?)	(:)	(;)
8-5	(↑)	(])	([)	(")
8-6	(')	(<)	(>)	(#)
8-7	(\\)	(!)	(&)	(%)

NOTE

Characters in parentheses indicate the ASCII equivalents in the DEC026 Character Set. The following punch configurations are sometimes associated with the following alternate graphic representations:

12	ampersand (&)
8-3	pound sign (#)
8-4	apostrophe (')
0-8-4	percent (%)
12-8-4	square box

Special characters:

ALTMODE	= 12-1-8 multiple punched
end-of-file	= All rows punched in column 1

10.7 OPERATION OF THE CR03B HANDLER

The CR03B card reader handler task operates in the same way as the CR15 handler task with the following exceptions.

1. To restart the CR03B after an error has been detected, depress the motor start and read start buttons instead of the power and reset buttons.

2. The error message:

   *** CD BAD DATA

   will be printed by the CR03B handler task whenever light check, dark check, itacher fail, or sync fail errors occur. To correct any of these errors remove the top card from the output itacher, place it at the bottom of the card deck in the input hopper, and restart the card reader.

3. The error message:

   **** CD DATA MISSED/PHOTO ERROR

   signifies the card reader received N column ready interrupts where N was not equal to 80. The cause and corrective action are identical to those for the CR15 handler with the exception noted in (1) above.

4. The source file for the CR03B handler task is named CR.03B nnn instead of CD.... nnn where nnn is the file's edit number.

CHAPTER 11

LINE PRINTER I/O

11.1 LP HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for the line printer is named LP.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is LP for purposes of reassignment.

There are no imposed page ejects. The user is free to count lines and print titles, or to construct continuous charts over form perforations.

The following line formats are supported:

1. IOPS ASCII. A two-word header, 5/7 packed text, and a carriage return, ALTMODE, or vertical control character line-terminator

2. IMAGE ALPHA. A two-word header, character-per-word (bits 11-17) text, and a carriage return, ALTMODE, or vertical control character line-terminator

3. OTS ASCII. A two-word header, an "OTS Control Character," 5/7 packed text, and a carriage return, ALTMODE, or vertical control character line terminator. The OTS Control Characters are:

Code	Meaning
12	Space and print
14	Eject page and print
20	Overprint previous line
21	Double space and print

Output in all data modes is accomplished by first moving the line from the requester's line buffer to a buffer within the Handler Task, because the line may have to be modified. The line buffer header word-pair-count is used to determine the line size and the checksum word is ignored. The request is considered completed once the line has been moved and before the line is actually printed.

Output in IOPS ASCII and IMAGE ALPHA is preceded by an up-space (line feed). Output in OTS ASCII requires that the handler modify the header to indicate two lines. This is necessary so that the "print multiple" IOT can be used and the OTS control character can be changed to a 15 code (carriage return) if it is a 20 code (overprint).

11.2 LP HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the LP I/O device handler task. Following are the legal function calls that can be issued for LP service, with a brief description of each:

Table 11-1
Line Printer I/O Functions

LP Call	Function
HINF	Request handler task information
ATTACH	Obtain exclusive use of line printer
DETACH	Release attached line printer
ABORT	Abort I/O for a task
WRITE	Write output from buffer
CLOSE	Relay CLOSE to the PDP-11 (UNICHANNEL version only)

HINF, ATTACH, DETACH and ABORT are basic I/O calls that operate almost exactly as described in Chapter 2. When requested by the LP handler task, however, HINF does require the special considerations described below.

11.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the event variable is set to +100011 (octal) to indicate the following:

Bit	Contents
0	Set to 0 to make the event variable positive
1-2	Set to 1 to indicate an output-only device
3	Set to 0 to indicate a non-directory-oriented handler
4-11	Set to device unit 0
12-17	Set to device code 11 (LP)

11.3 WRITE: WRITING OUTPUT FROM A BUFFER

The WRITE I/O function causes the Handler to print a line. It is unnecessary to attach before WRITing. WRITE operates almost exactly as described in Chapter 2. Its CPB has the following form:

Word	Contents
0	2700 (I/O function code)
1	Event Variable address
2	Logical Unit number
3	I/O data mode (2 indicates ASCII; 3 indicates IMAGE)
4	Line header address

11.4 UC15 OPERATION

The UC15 printers are programmed exactly like the XVM printer. The primary difference is the presence of the spooling feature in the PDP-11. When this feature is enabled (with the DOS SPOOL function), print lines proceed to the line printer via the RK05 disk. To the program, the line printer appears to run at disk speed. The PDP-11 then prints out the accumulated information on the disk. New print information may be added while a previous job is still printing.

In case of a line printer error, the POLLER will print the following on LUN-3:

```
*** UC15 ERROR LPU 000004
```

To obtain a UC15 Handler, the assembly parameter UC15=0 must be specified. If desired, the spooling feature may be disabled by assembling with the parameter NOSPL=0.

CHAPTER 12

PAPER TAPE READER I/O

12.1 PR HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for the paper tape reader is called PR.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is PR for purposes of reassignment.

NOTE

Tapes punched on model ASR Teletypes typically are the oil-base, non-fanfold variety. Such tapes conduct light too easily and cannot be reliably read photoelectrically.

12.2 PR HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the PR I/O Device Handler Task. Following are legal function calls which may be issued for PR service and a brief description of each:

Table 12-1
Paper Tape Reader I/O Functions

PR Call	Function
HINF	Request Handler Task information
ATTACH	Obtain exclusive use of paper tape reader
DISCONNECT & EXIT	Disconnect Handler from interrupt line and exit
DETACH	Release attached paper tape reader
READ	Read input into buffer
CLOSE	Unload tape
ABORT	Abort I/O for Task

HINF, ATTACH, DETACH, and DISCONNECT & EXIT are basic I/O calls and operate almost exactly as described in Chapter 2. When requested by the PR Handler Task, however, HINF does require the few special considerations discussed below.

12.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +200006 (octal) to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 2 to indicate input-only device
3	Set 0 to indicate a non-directory-oriented Handler
4-11	Device unit 0
12-17	Device code 6 (paper tape reader)

12.3 READ: READING INPUT INTO A BUFFER

The READ I/O function call READs input into a buffer via the PR Handler Task assigned to the specified Logical Unit Number. It operates almost exactly as in the description of READ in Chapter 2, with the exceptions and special features described below.

All tape frames are read one at a time regardless of the data mode. When the physical end of a tape is reached, a hardware flag is raised which is also set when the reader OFF LINE switch is pushed.

Since the two conditions cannot be distinguished from one another, end-of-tape is transparent to the requesting program. Whenever end-of-tape or OFF LINE is detected, the following message is printed on the MCR terminal (LUN-3):

```
*** LOAD PAPER READER (OFF LINE)
```

The operator is requested to load another tape in the reader. He must have the reader switch in the OFF LINE position. Once this has been done and the reader switch is set in the ON LINE position, the tape begins to move immediately. (The Handler uses a MARK-time Directive to give up control for a brief interval, and when it regains control it rechecks the reader status.) The primary use of the OFF LINE switch is to suspend reading so that portions of a large tape file can be easily removed or inserted.

Two minor drawbacks result from not terminating a READ when out-of-tape condition arises:

1. It is difficult to write a simple paper tape copy program since the program has no way of determining the length of a tape without resorting to a cue from the operator.

2. A data frame is read as the trailing edge of the tape passes through the reader. If this erroneous frame is blank, it is ignored in the IOPS data modes; however, it is possible to receive a nonzero frame. For this reason, files should not be segmented onto several tapes unless the Task that is reading the tapes can utilize the CLOSE function to avoid the problems of transition between two tapes.

If the size of the requester's line buffer is larger than 776, it is taken to be 776 (twice 377) since 377 is the largest word-pair-count that can be recorded in the line buffer header.

12.3.1 IOPS BINARY (Mode 0)

This section summarizes characteristics of IOPS BINARY mode relevant to paper tape input.

1. All IOPS BINARY tape frames are punched in odd parity with bit 8 always set. All frames which do not have the eighth bit set are ignored. A parity test is made only on data frames that enter the requester's line buffer; this excludes ignored tape frames and excess data (see below).

2. Three binary tape frames constitute one binary word.

3. IOPS BINARY is the only data mode in which a header word pair is punched on paper tape. Therefore, the buffer size given by the READ request limits, but does not specify, the number of data words to be read from the tape. The first word read from tape is header word 0. The word-pair-count must be greater than zero, the data validity bits must be zero, and the data mode must be zero (IOPS BINARY). Bits 9-11 are ignored since they may be nonzero if the tape was punched in the Advanced Software System.

   If the record size, which is given by the word-pair-count, is larger than the line buffer, the data validity bits in line buffer header word 0 are set to indicate a short buffer error. The buffer is then filled and the words remaining in the tape record are declared excess data, which are simply read and ignored.

4. As data words are entered in the line buffer, they are added together to form a checksum, which should be zero when the record has been read.

5. The data validity bits (12, 13) in header word 0 can only indicate one type of error. If multiple errors occur during a READ, the line buffer header is set in order of precedence as follows: parity error, then short line, and finally checksum error. Even if these three errors occur, the requester's Event Variable is set to +1 to indicate completion of the I/O request.

12.3.2 IMAGE BINARY (Mode 1)

This section summarizes characteristics of IMAGE BINARY mode relevant to paper tape input.

1. IMAGE BINARY tape frames are punched with bit 8 always set. All frames which do not have the eighth bit set are ignored.

2. Three binary tape frames constitute one binary word.

3. In this data mode, the tape has no header word pair. Therefore, the record size is determined by the size of the line buffer. Because no end-of-file, end-of-tape, or end-of-medium condition can be returned to the requester, the requesting program must have information on the number of data words on the tape.

4. No error conditions are flagged in the data validity bits of line buffer header word 0 since parity and checksum tests are not made. The left half of header word 0 is set with the word-pair-count (computed from the buffer size), and the right half is set to mode 1 (IMAGE BINARY). The contents of header word 1 (the checksum word) are left unchanged.

12.3.3 IOPS ASCII (Mode 2)

This section summarizes characteristics of IOPS ASCII mode relevant to paper tape input.

1. IOPS ASCII tape frames contain 7-bit ASCII characters with the eighth bit set so that the frame parity is even (as opposed to binary tapes).

2. Since no header word pair is punched on the tape, the line buffer size is used initially as the record size. This count is truncated to an even number because IOPS ASCII packs data in word-pairs. Input to the line buffer is normally terminated when an IOPS ASCII line terminator character (carriage return or ALTMODE) is read from the tape.

3. If the character codes 33 or 176 are read, they are converted to 175, which is the standard internal representation for ALTMODE.

4. Null (0) and rubout (177) characters are discarded. Nulls form the blank leader and trailer on a tape and are used as timing characters following horizontal tab, vertical tab, and form feed. Rubout may also be used for timing in this way, and in off-line tape preparation it is punched over characters that were mistakes (it is difficult to erase holes in a paper tape).

5. The character 04, representing CTRL/D or EOT (8-bit code 204), is special. It is treated as an end-of-file marker in lieu of an end-of-file header. So that tapes may be listed off-line, headers are not punched in IOPS ASCII.

   If this character is read, an end-of-file flag is set (see below) and the character is converted into a carriage return (15).

6. If a character is not ignored as excess data (see below), its parity is checked. Odd parity is an error, but the character is still passed on to the requester and reading continues.

7. If a line terminator (carriage return or ALTMODE) is not encountered when the end of the line buffer is reached, a short buffer error condition is flagged in the data validity bits of header word 0 and the last character in the buffer is changed to a carriage return as a precaution. It is assumed that the requester's line buffer is too short -- the tape must be positioned to the beginning of the next line. Characters continue to be read from the tape, but are treated as excess data and are discarded.

8. When input is terminated, header word 0 is set as follows: the word-pair-count is stored in the left half; the mode is set to 5 if end-of-file occurred, otherwise it is set to 2 to indicate IOPS ASCII; the data validity bits are set if an error occurred. A parity error takes precedence over a short buffer error.

9. Checksum is not computed. The checksum word (header word) is set to zero.

12.3.4 IMAGE ASCII (Mode 3)

This section summarizes characteristics of IMAGE ASCII mode relevant to paper tape input.

1. IMAGE ASCII tape frames contain 8-bit data. Therefore, there is no required parity, and the eighth bit has no special characteristics.
2. Each 8-bit character is stored in one line buffer word.
3. The record size is determined directly from the line buffer size and this determines the word-pair-count set in header word 0. The tape itself has no header word pair. The mode in header word 0 is set to 3 to indicate IMAGE ASCII and the data validity bits are set to 0 since no error checking is done.
4. Checksum is not computed and the checksum word (line buffer header word 1) is unaltered.
5. The program must know how much data to read because IMAGE ASCII mode provides no way to create an end-of-file and because the end-of-tape condition is not passed on to the requesting program. Note that null (blank) tape frames are passed on to the line buffer in this data mode.

12.4 CLOSE: CLOSING A FILE AND UNLOADING TAPE

CLOSE is associated with directory-oriented devices. It is a declaration that there are to be no further references (data transfers) to or from the current file and that therefore the file is to be CLOSEd. CLOSE operates almost exactly as described in Chapter 2.

Each paper tape should be treated as a file. The CLOSE function causes the remainder of the tape to pass through the reader. Tape frames are ignored. This function is not simply a convenience to the operator. It prevents a Task from starting a read operation at the trailer end of the last tape. A nonzero frame might be read as the trailing edge of the tape passes through the reader.

12.5 ABORT: ABORTING I/O FOR A TASK

The Handler for a high-speed device can ignore a pending ABORT request until it finishes the current I/O operation. The reader Handler cannot ignore a pending ABORT. It may process the ABORT in two situations:

1. When I/O is not in progress and the ABORT node is the next request to be dequeued

2. When the Handler has continued execution following an "I/O done" response from its interrupt service routine, from the TTY Handler, or at completion of a mark-time request

In the latter case, I/O is in progress, but not necessarily for the Task being ABORTed. The ABORT is honored as described below and then the I/O that was in progress is resumed, provided it was for a different Task.

The following is done for the ABORT function:

1. If the Task has attached the device, a DETACH is performed.

2. Any I/O requests made by that Task which remain in the I/O request queue are removed, and the nodes are returned to the Pool of Empty Nodes.

CHAPTER 13

PAPER TAPE PUNCH I/O

13.1 PP HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for the paper tape punch is named PP.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is PP for purposes of reassignment.

13.2 PP HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the PP I/O Device Handler Task. Following are legal function calls which may be issued for PP service and a brief description of each:

Table 13-1
Paper Tape Punch I/O Functions

PP Call	Function
HINF	Request Handler Task information
ATTACH	Obtain exclusive use of paper tape punch
DETACH	Release attached paper tape punch
DISCONNECT & EXIT	Disconnect Handler from interrupt line and exit
WRITE	Write output from buffer
CLOSE	Close file
ABORT	Abort I/O for Task

HINF, ATTACH, DETACH, and DISCONNECT & EXIT are basic I/O calls and operate almost exactly as described in Chapter 2. When requested by the PP Handler Task, however, HINF does require the few special considerations discussed below.

13.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +100010 (octal) to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 1 to indicate an output-only device
3	Set 0 to indicate a non-directory-oriented Handler
4-11	Device unit 0
12-17	Device code 10 (paper tape punch)

13.3 WRITE: WRITING OUTPUT FROM A BUFFER

The WRITE I/O function call WRITEs output from a buffer to the PP Handler Task assigned to the specified Logical Unit Number. It operates almost exactly as in the description of WRITE in Chapter 2, with the exceptions and special features discussed below.

All tape frames are punched one at a time regardless of data mode. When only one inch of tape remains, an out-of-tape flag is raised but does not cause a program interrupt. Therefore, the tape status is checked before each frame is punched.

If the punch is out of tape, the following message is output to Logical Unit 3 (the MCR terminal):

```
*** LOAD PAPER PUNCH. THEN, "RESUME PP...."
```

The punch Handler then issues a SUSPEND Directive. The operator must reload the punch and request resumption of punch Handler execution.

Since tape frames are punched one at a time, the paper tape reader Handler reads binary records one frame at a time and binary words may be split between two tapes. However, files should not be segmented this way onto several paper tapes. The trailing edge of a tape may be mistakenly read as an erroneous data frame.

For the very first WRITE request and for the first WRITE request following a CLOSE, two fanfolds of blank tape are punched to form a leader.

13.3.1 IOPS BINARY (Mode 0)

This section summarizes characteristics of IOPS BINARY mode relevant to paper tape output.

1. All IOPS BINARY tape frames (3 per binary word) are punched in odd parity with bit 8 always set to one.

2. IOPS BINARY is the only data mode in which a line buffer header word pair is punched on paper tape. The right half of header word 0 (the data validity bits and the mode indicator) is set to zero by the punch Handler. Header word 1 (the checksum word) is computed and set by the punch Handler so that the two's complement sum of all the words in the record (including the header word pair) is zero.

3. The word-pair-count in the line buffer header determines the size of the punched record. A single blank tape frame separates binary records from one another. The blank frame is ignored when read and acts only as a visual record separator.

13.3.2 IMAGE BINARY (Mode 1)

This section summarizes characteristics of IMAGE BINARY mode relevant to paper tape output.

1. All IMAGE BINARY tape frames (3 per binary word) are punched in odd parity with bit 8 always set to one. The parity is not required in this mode and is ignored by the paper tape reader Handler. It is computed and punched because both binary modes share common code in the paper tape punch Handler. This means that read-in-mode tapes cannot be punched in this mode.

2. The word-pair-count specified in the line buffer header determines the record size, but the line buffer header is not punched. A single blank tape frame separates binary records from one another. The blank frame is ignored when read and acts only as a visual record separator.

13.3.3 IOPS ASCII (Mode 2)

This section summarizes characteristics of IOPS ASCII mode relevant to paper tape output.

1. IOPS ASCII tape frames contain 7-bit ASCII characters with the eighth bit set so that the frame parity is even (as opposed to binary tapes).

2. Because no header is punched in this mode, the tape may be listed off-line if desired.

3. The first character in the line buffer is checked to see if it is a line feed or overprint. If it is neither, a line feed is punched before the buffer is punched. The line feed is not substituted for the first character in the buffer.

   This makes off-line tape listing possible. It is typical for ASCII lines output by MACRO programs to end with a carriage return and not to include a vertical form control character.

4. Again, to facilitate off-line tape listing, timing (or filler) characters are punched following horizontal tab, vertical tab, and form feed. Two nulls are punched after a horizontal or a vertical tab, and ten nulls are punched after a form feed. These fillers provide the timing delay that is necessary to ensure the completion of the carriage control function before the next character is printed. They give adequate protection for listings on 110-baud terminals.

5. Punching of an IOPS ASCII line normally terminates when a carriage return or ALTMODE character is encountered. The word-pair-count given in the line buffer header is used as a limit factor. If the entire buffer is punched but no line terminator is found, a carriage return is punched and the line is thereby terminated.

13.3.4 IMAGE ASCII (Mode 3)

This section summarizes characteristics of IMAGE ASCII mode relevant to paper tape output.

1. IMAGE ASCII tape frames consist of 8-bit data completely specified by the user; hence the Handler does not compute parity.

2. The word-pair-count given in the line buffer header determines the number of frames to be punched, but no header word pair is output to the tape. One tape frame is punched from bits 10 through 17 of each data word in the line buffer.

3. This data mode gives the user complete control over what is punched on a tape.

13.4 CLOSE: CLOSING A FILE

The CLOSE I/O function call informs the PP Handler Task assigned to the specified Logical Unit Number that the issuing Task has completed a set of related I/O operations on the current file. CLOSE operates almost exactly as described in Chapter 2, with the exceptions and special features discussed below. Independent of the most recent data mode, CLOSE causes two fanfolds of blank tape (a "trailer") to be punched.

13.4.1 IOPS BINARY (Mode 0)

If the data mode associated with the most recent WRITE request is IOPS BINARY, an end-of-file record consisting of two words is punched:

```
001005

776773
```

Word 0 indicates a word-pair-count of one and gives the end-of-file indicator (5) in the mode bits. Word 1, the checksum word, is simply the two's complement of Word 0.

13.4.2 IOPS ASCII (Mode 2)

This section summarizes characteristics of IOPS ASCII mode relevant to closing a file after punching. If the data mode associated with the most recent WRITE request is IOPS ASCII, a pseudo-end-of-file is punched as follows:

```
8 null tape frames

204
```

The null tape frames are provided for visual separation and make it easy to remove the 204 from the tape. ASCII code 04 representing CTRL/D or end-of-transmission (8-bit code 204), is treated by the paper tape reader Handler as an end-of-file character (when reading in IOPS ASCII mode only). The reader Handler does not pass the code 04 to the requester's line buffer. Instead, it stores a carriage return and sets the mode bits in the line buffer header to 5, which means end-of-file.

Use of code 04 in this way is necessary if IOPS ASCII files are to be punched on several paper tapes (segmented). This is so because, in RSX, the fact that a file is segmented onto several tapes is transparent to the program reading the file. But in the Advanced Software System, the end-of-tape (end-of-medium) condition is passed on to the requesting program. Consequently a code 04 is not punched at the end of an IOPS ASCII file when prepared with ADSS.

13.4.3 IMAGE BINARY and IMAGE ASCII (Modes 1 and 3)

For IMAGE BINARY and IMAGE ASCII modes, CLOSE does not punch an end-of-file. Consequently, the length of the file cannot be determined by a program which does not have information on the data encoding scheme (e.g., PIP in ADSS).

13.5 ABORT: ABORTING I/O FOR A TASK

The Handler for a high-speed device need not respond to a pending ABORT request until it finishes the current I/O operation. The punch Handler cannot ignore a pending ABORT. It may process the ABORT in three situations:

1. When I/O is not in progress and the ABORT node is the next request to be dequeued

2. When the Handler has continued execution following an "I/O done" response from its interrupt service routine or from the TTY Handler

3. When the Handler has resumed execution following its SUSPEND for an out-of-tape condition

In cases 1 and 2, I/O is in progress, but not necessarily for the Task being ABORTed. The ABORT is honored as described below and then the I/O that was in progress is resumed, provided it was for a different Task.

The following is done for the ABORT function:

1. If the Task has attached the device, a DETACH is performed.

2. Any I/O requests made by that Task which remain in the I/O request queue are removed, and the nodes are returned to the Pool of Empty Nodes.

3. If I/O is not in progress for a different Task, a flag is reset so that the next Task to use the punch will get two fanfolds of leader.

CHAPTER 14

ANALOG-TO-DIGITAL CONVERTER I/O

The AD Analog-to-Digital Converter is a special-purpose device which runs under RSX and facilitates the conversion of analog signals to a digitized format. To program this device properly, the user must be intimately aware of the operational characteristics of the hardware peripherals and of the software I/O Handler.

14.1 AD HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for the AD converter is named AD.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is AD for purposes of reassignment. It allows protected and privileged Tasks to read digitized analog signals from the AD Analog Subsystem. The number of signals that can be converted per QUEUE I/O request is an important consideration, since the AD converter permits a maximum of 25,000 conversions per second (40 microseconds per conversion). This implies that the user Task must have a large enough buffer to accommodate these signals or that the Handler itself can provide a special facility to allow multibuffering without loss of conversions. Two special-purpose I/O function calls, ADRSET and ADSSET, facilitate this multibuffering requirement and are described in this chapter.

The AD.... Handler Task must be built to run in EXEC mode at priority 1.

14.2 AD HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the AD I/O Device Handler Task. Following are the legal function calls which may be issued for AD service and a brief description of each:

Table 14-1
AD Converter I/O Functions

AD Call	Function
HINF	Request Handler Task information
ABORT	Abort I/O for a Task
DISCONNECT & EXIT	Disconnect the Handler from interrupt line and exit
ADCON	Connect AD Handler to Task
ADDIS	Disconnect AD Handler from Task
ADSSET	Establish AD sequential data-channel mode
ADRSET	Establish random data-channel mode
ADSTRT	Start AD conversion
ADSTOP	Stop AD conversion

In addition, two FORTRAN-callable subroutines, ADSMAP and ADRMAP, have been implemented to map input status parameters into an appropriate internal format before starting sequential and random AD conversion respectively. All functions and subroutines are described in subsequent sections.

HINF, ABORT, and DISCONNECT & EXIT are basic I/O function calls and operate almost exactly as described in Chapter 2. When requested by the AD Handler Task, however, HINF does require the few special considerations discussed below.

14.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +200020 to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 2 to indicate input-only device
3	Set 0 to indicate a non-directory-oriented Handler
4-11	Device unit 0
12-17	Device code 20 (AD Converter)

14.2.2 Significant Event Declaration

When a USER-mode Task is connected, AD.... does not declare a Significant Event each time an AD interrupt occurs. Instead, it sets register 146 (octal) of the Executive so that a clock-generated Significant Event occurs after a certain number of clock ticks (146 octal normally contains 60 decimal). The number of clock ticks is an assembly parameter with a default of 1. When the AD Handler Task exits, or when the connected USER-mode Task disconnects or is aborted, the previous contents of 146 are restored. This is done to prevent the rate of Significant Event declaration from climbing so high that it would shut out the rest of the system.

The above precaution is not observed for EXEC-mode Tasks. If the Task connected to the AD Handler is built in EXEC mode, every AD interrupt causes declaration of a Significant Event.

14.3 ADCON: CONNECTING AD TO A TASK

The ADCON I/O function call connects the AD Handler to the Task which requests it. Acceptance of the ADCON request results in initializing the link table and establishing exclusive Task use of the AD Handler. ADCON must be issued before any other AD request can be accepted. It is used instead of ATTACH, because ATTACH implies that more than one Task might be able to use the AD.... Handler if it were detached.

ADCON can be issued by a FORTRAN program in the following format:

Form:	CALL ADCON (LUN[,ev])
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable
Example:	CALL ADCON (LUN,IEV)

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	0500 (I/O function code)
1	Event Variable address
2	Logical Unit Number

14.4 ADDIS: DISCONNECTING AD FROM A TASK

The ADDIS I/O function call disconnects the AD Handler from the requesting Task. This frees the Handler for use by other Tasks.

ADDIS can be issued by a FORTRAN program in the following format:

Form:	CALL ADDIS (LUN[,ev])
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable
Example:	CALL ADDIS (LUN,IEV)

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	0600 (I/O function code)
1	Event Variable address
2	Logical Unit Number

14.5 ADSSET: ESTABLISHING SEQUENTIAL LINKS

The ADSSET I/O function call sets up an I/O parameter link table within the AD Handler for subsequent sequential data-channel mode operation. A table consists of ten (decimal) links. ADSSET can be issued by a FORTRAN program in the following format:

Form:	CALL ADSSET (LUN,num,type,lch,uch,isw,dvt[, rep[,lev[,moev[,ev]]]])
Where:	LUN is decimal and represents the Logical Unit Number num is the link number (1-10 decimal) type is the link type (0-3) lch is the lower channel number (0-127 decimal) uch is the upper channel number (0-127 decimal) isw is the input status word address dvt is the first element of the digital values table rep is the repeat count (if no nonzero value is specified, 1 is assigned) lev is the integer Link Event Variable moev is the integer Memory Overflow Event Variable ev is the integer QUEUE I/O Event Variable
Example:	CALL ADSSET (LUN,LKNO,LKTYPE,LCH,UCH,ISW, IDVTE,ICNT,LEV,IMOEV,IEV)

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	0100 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table is constructed as follows:

Word	Contents
0	Link number (1-10 decimal)
1	Link type (0, 1, 2, or 3)
2	Lower channel number (0-127 decimal)
3	Upper channel number (0-127 decimal)
4	Input status word address
5	Digital values table address
6	Repeat count (default value 1)
7	Link Event Variable
10	Memory Overflow Event Variable

The link type (word 2) may contain one of the following:

Type	Meaning
0	Null link; ignore during I/O processing
1	Chain link; after this link is processed, process the next link in the AD Handler Task link table
2	End link; after this link is processed, stop I/O
3	Loop link; after this link is processed, go to head of link table and process the next link

If no end link is encountered, end-around looping on the link table is performed. If all links are null, no I/O processing is initiated.

For sequential mode conversions, only a single input status word (word 4) is used. It has the following bit designations:

Bit	Contents
0-1	Gain (00 = 1, 01 = 2, 10 = 4, 11 = 8)
2-5	Unused
6	Memory overflow (1 = enabled)
7	Data channel break (1 = enabled)
8	Internal/external sync (1 = external)
9	Add-to-memory mode (1 = enabled)
10	Data channel operation (1 = enabled)
11-17	Analog channel address

The starting analog channel number (bits 11-17) is set by ADSSET processing. A FORTRAN-callable subroutine called ADSMAP is responsible for mapping parameters into input status form for the FORTRAN programmer using sequential conversion.

Once I/O has been initiated (via the ADSTRT function), the Link Event Variable (word 7) is set to one plus the number of repetitions remaining for the specified link number (word 0) following each transfer.

14.6 ADRSET: ESTABLISHING RANDOM LINKS

The ADRSET I/O function call sets up an I/O parameter link within the AD Handler for subsequent random data-channel mode operation. A table consists of ten (decimal) links. ADRSET can be issued by a FORTRAN program in the following format:

Form:	CALL ADRSET (LUN,num,type,pts,stat,dvt[,rep[, lev[,moev[,ev]]]])
Where:	LUN is decimal and represents the Logical Unit Number num is the link number (1-10 decimal) type is the link type (0-3) pts is the number of points to convert stat is the first element of the input status table dvt is the first element of the digital values table rep is the repeat count (if no nonzero value is specified, 1 is assigned) lev is the integer Link Event Variable moev is the integer Memory Overflow Event Variable ev is the integer QUEUE I/O Event Variable
Example:	CALL ADRSET (LUN,LKNO,LKTYPE,NPTS,ISTEI, IDVTE,ICNT,LEV,MOEV,IEV)

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	0200 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table is constructed as follows:

Word	Contents
0	Link number (1-10 decimal)
1	Link type (0, 1, 2, or 3)
2	Number of conversions
3	Input status table address
4	Digital values table address
5	Repeat count (default value 1)
6	Link Event Variable
7	Memory Overflow Event Variable

The link type (word 2) may contain one of the following:

Type	Meaning
0	Null link; ignore during I/O processing
1	Chain link; after this link is processed, process the next link in the AD Handler Task link table
2	End link; after this link is processed, stop I/O
3	Loop link; after this link is processed, go to head of link table and process the next link

If no end link is encountered, end-around looping on the link table is performed. If all links are null, no I/O processing is initiated.

With ADRSET there is a one-to-one correspondence between the Input Status Table and the Digital Values Table. The bit assignments for words in the Input Status Table are as follows:

Bit	Contents
0-1	Gain (00 = 1, 01 = 2, 10 = 4, 11 = 8)
2-5	Unused
6	Memory overflow (1 = enabled)
7	Data channel break (1 = enabled)
8	Internal/external sync (1 = external)
9	Add-to-memory mode (1 = enabled)
10	Data channel operation (1 = enabled)
11-17	Analog channel address

A FORTRAN-callable subroutine called ADRMAP is responsible for mapping parameters into input status form for the FORTRAN programmer using random conversion.

Once I/O has been initiated (via the ADSTRT function), the Link Event Variable (word 6) is set to one plus the number of repetitions remaining for the specified link number (word 0) following each transfer.

14.7 ADSTRT: STARTING AD CONVERSION

The ADSTRT I/O function call starts conversion by the AD Analog-to-Digital Converter by initiating processing of the Handler's link table.

ADSTRT can be issued by a FORTRAN program in the following format:

Form:	CALL ADSTRT (LUN[,ev])
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable
Example:	CALL ADSTRT (LUN,IEV)

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	0300 (I/O function code)
1	Event Variable address
2	Logical Unit Number

ADSTRT actually initiates I/O transfers, while ADSSET and ADRSET set up the AD Handler's link table which controls I/O transfers at the interrupt level.

14.8 ADSTOP: STOPPING AD CONVERSION

The ADSTOP I/O function call stops conversion by the AD Converter by terminating processing of the Handler's link table. Processing will terminate after completion of the next I/O transfer.

ADSTOP can be issued by a FORTRAN program in the following format:

Form:	CALL ADSTOP (LUN[,ev])
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable
Example:	CALL ADSTOP (LUN,IEV)

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	0400 (I/O function code)
1	Event Variable address
2	Logical Unit Number

14.9 ADSMAP: MAPPING SEQUENTIAL INPUT PARAMETERS

ADSMAP is a FORTRAN-callable subroutine used to map input status parameters into the proper internal form prior to starting sequential conversion. ADSMAP is not an actual I/O function, but its subroutine call takes the following form:

Form:	CALL ADSMAP (isw,gain,error[,addmem[,sync [,mov]]])
Where:	isw is the input status word gain is the integer gain value (1, 2, 4, or 8) error is the integer error variable set nonzero on error addmem is the integer add-to-memory variable (nonzero value enables add-to-memory; zero value (default) disables add-to-memory) sync is an integer variable (nonzero value signifies external sync; zero value (default) signifies internal sync mov is an integer variable (nonzero value disables memory overflow; zero value (default) enables memory overflow)

14.10 ADRMAP: MAPPING RANDOM INPUT PARAMETERS

ADRMAP is a FORTRAN-callable subroutine used to map input status parameters into the proper internal form prior to starting random conversion. ADRMAP is not an actual I/O function, but its subroutine call takes the following form:

Form:	CALL ADRMAP (stat,gain,chan,error,[,addmem[, sync[,mov]]])
Where:	stat is an element in the input status table corresponding to the appropriate analog channel gain is the integer gain value (1, 2, 4, or 8) chan is the integer analog channel number error is the integer error variable set nonzero on error addmem is the integer add-to-memory variable (nonzero value enables add-to-memory; zero value (default) disables add-to-memory) sync is an integer variable (nonzero value signifies external sync; zero value (default) signifies internal sync) mov is an integer variable (nonzero value disables memory overflow; zero value (default) enables memory overflow)

CHAPTER 15

AUTOMATIC FLYING CAPACITOR SCANNER I/O

The AFC Automatic Flying Capacitor Scanner is a special-purpose front-end device which runs under RSX. The AFC is a differential analog input device for industrial data-acquisition control systems. While minimizing noise, it multiplexes differential input analog signals, selects gain, and performs analog-to-digital conversion. To program it properly, the user must be intimately aware of the operational characteristics of the hardware peripherals and of the software I/O Handler.

15.1 AF HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for the AFC Scanner is named AF.... and handles calls submitted through special forms of the QUEUE I/O Directive. Requests are queued by requester priority and dequeued either by priority or by Task, depending on whether a Task has attached the device. The AFC Scanner is considered a single-unit device whose name is AF for Logical Unit assignment purposes.

The AF Handler Task must be assembled with the parameters FMAD and NMOD defined. FMAD is a constant defining the address of the first module in bits 0 through 7, and NMOD is a constant defining the number of AFC modules. For hardware purposes, the module address is considered to have three parts: X in bits 0 through 2, Y in bits 3 through 5, and WD in bits 6 through 7.

15.2 AF HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the AF I/O Device Handler Task. The Logical Unit Number (LUN) is assigned to AF. Following are the legal function calls that may be issued for AF service and a brief description of each:

Table 15-1
AFC I/O Functions

AF Call	Function
HINF	Request Handler Task information
ATTACH	Obtain exclusive use of AFC Scanner
DETACH	Release attached AFC Scanner
AI	Queue request to read sequence of analog channels

They are basic I/O calls and operate almost exactly as described in Chapter 2. When requested by the AF Handler Task, however, HINF does require the few special considerations discussed below.

15.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +200016 (octal) to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 2 to indicate input-only device
3	Set 0 to indicate non-directory-oriented Handler
4-11	Device unit 0
12-17	Device code 16 (AFC Scanner)

15.3 AI: READING A SEQUENCE OF ANALOG CHANNELS

The AI function (which is a type of GET function) is used to read a sequence of analog input channels on the AFC Automatic Flying Capacitor Scanner. Because an A/D conversion is performed in 5ms, and a channel may not accurately be re-read within 50ms, the Handler Task inserts a delay when necessary to prevent re-reading a channel too soon. To facilitate this delay, channel 2047 is dedicated for Handler use.

AI is only issued by a FORTRAN program in the following way:

Form:	CALL AI (LUN,fchan,lchan,gain,data[,ev])
Where:	LUN is decimal and represents the Logical Unit Number fchan is the first channel number (from one) lchan is the last channel number (from one) gain is the integer Gain Table array, initialized with gain-per-channel information data is the integer Data Table array which receives digitized values ev is the integer Event Variable

Example:	Read analog input channels 17 through 26 with a gain of 1000, and output the six digital values to LUN-10:
	`DIMENSION IGA(6),IDA(6)`
	`.`
	`.`
	`.`
	`DO 100 J=1,6`
	`100 IGA(J)=1000`
	`CALL AI(10,17,26,IGA,IDA,IEV)`
	`CALL WAITFR(IEV)`
	`WRITE(10,200)IDA`
	`200 FORMAT (6I4)`

Channel fchan is multiplied by gain (1), digitized, and stored in data (1).

The CPB for this form of QUEUE I/O has the following form:

Word	Contents
0	3000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table is constructed as follows:

Word	Contents
0	First channel number (from zero)
1	Last channel number
2	Gain Table address
3	Data Table address

The information from the Control Table is not queued along with the information from the CAL Parameter Block. Therefore, after making the I/O request, one must not modify the Control Table until after completion of the request.

CHAPTER 16

UNIVERSAL DIGITAL CONTROLLER I/O

The UDC Universal Digital Controller is a special-purpose front-end device which runs under RSX. The UDC operates as a high-level digital multiplexer, interrogating digital inputs and driving digital outputs located on directly addressable functional modules. To program this device properly, the user must be intimately aware of the operational characteristics of the hardware peripherals and of the software I/O Handler.

16.1 UD HANDLER TASK

The I/O Device Handler Task that services requests for the UDC is named UD.... and handles calls submitted through special forms of the QUEUE I/O Directive. Requests are queued by requester priority and dequeued "off the top." The Universal Digital Controller is considered a single-unit device whose name is UD for Logical Unit assignment purposes.

Because of the generality of the UDC hardware, it is normally necessary to edit the Handler Task source to indicate the module-types to be serviced and their positions (addresses) in the UDC. The Handler is coded (by conditional assembly) so that code to support nonexistent modules may be eliminated.

A "module-address table" is defined for each module type this Handler Task supports. Modules are numbered from zero in the order of the module address table. Channels and points are numbered from zero, starting with module zero. The present set of UDC modules contains four analog channels per D/A module, and sixteen discrete points per digital input and output module.

16.2 UD HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the UD I/O Device Handler Task. The Logical Unit Number (LUN) is assigned to UD. Following are the legal function calls which may be issued for UD service and a brief description of each:

Table 16-1
UDC I/O Functions

UD Call	Function
HINF	Request Handler Task information
AO	Set analog output channel voltage
DOS	Pulse digital output point
DOL	Set digital output point
DI,(RBIN, RBCD,RDP)	Read contact sense/interrupt module
CTDI	Connect buffer for digital input
DFDI	Disconnect connected buffer

In addition, a FORTRAN-callable subroutine, RDDI, has been implemented to read digital input and clear the Trigger Event Variable. All functions and subroutines are described in subsequent sections. HINF is a basic I/O call and it operates almost exactly as described in Chapter 2. When requested by the UD Handler Task, however, it does require the few special considerations discussed below.

16.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +300017 (octal) to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 3 to indicate input and output device
3	Set 0 to indicate non-directory-oriented Handler
4-11	Device unit 0
12-17	Device code 17 (UDC)

16.2.2 FORTRAN Interface

In the FORTRAN calls described below, the Event Variable is optional. It is necessary, however, when request completion information is needed, because control is returned to the caller after a request is queued and often before completion of the requested function.

To maintain consistency with the FORTRAN convention of counting from one rather than zero, modules, channels, and points are numbered from one. Module, channel, and point numbers are thus one greater than those used to request UDC service from an assembly language program.

16.3 AO: ANALOG OUTPUT -- A633 MODULES

The AO I/O function call sets an indicated analog output channel to an indicated voltage.

AO is only issued by a FORTRAN program in the following way:

Form:	CALL AO (LUN,channel,volt[,ev])
Where:	LUN is decimal and represents the Logical Unit Number channel is decimal and represents the output channel number volt is the integer output voltage representation (0-1023 decimal represents 0-10 volts) ev is the integer Event Variable
Example:	Set channel 2 to 5 volts and, when it is set, zero channel 3: IVOLTS=5.0*102.4-1. CALL AO(32,2,IVOLTS,IEV) CALL WAITFR(IEV) CALL AO(32,3,0)

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	6700 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Output channel number
4	Output voltage representation

16.4 DOS: DIGITAL OUTPUT, SINGLE-SHOT -- M687 AND M807 MODULES

The DOS I/O function call pulses (closes for a preset time interval) an indicated digital output point. DOS is only issued by a FORTRAN program in the following way:

Form:	CALL DOS (LUN,point[,ev])
Where:	LUN is decimal and represents the Logical Unit Number point is decimal and represents the digital point number ev is the integer Event Variable
Example:	Pulse single-shot point 3: CALL DOS(32,3,IEV)

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	6000(I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Digital point number

16.5 DOL: DIGITAL OUTPUT LATCHING -- M685,M803,AND M805 MODULES

The DOL I/O function call sets an indicated digital output point to an indicated logical value. A logical value of .TRUE. implies CONTACTS CLOSED and is represented by a word with all bits set on. A logical value of .FALSE. implies CONTACTS OPEN and is represented by a word with all bits cleared (set 0). DOL is only issued by a FORTRAN program in the following way:

Form:	CALL DOL (LUN,point,var[,ev])
Where:	LUN is decimal and represents the Logical Unit Number point is decimal and represents the digital point number var is the logical variable ev is the integer Event Variable

Example:	Close (short) latching output points 10-19:
	`      DO 100 IDP = 10,19`
	`100   CALL DOL (32,IDP,.TRUE.,IEV)`

The CPB for this form of QUEUE I/O follows:

Word	Contents
0	6100 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Digital point number
4	Logical variable

16.6 DI(RBIN,RBCD,RDP): CONTACT SENSE/INTERRUPT DIGITAL INPUT -- W731 AND W733 MODULES

The DI I/O function call reads a contact sense or contact interrupt module. The CPB has the following form:

Word	Contents
0	7400 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Module number
4	Module data word address

The module data are returned in the low-order 16 bits of word 4.

A contact sense or contact interrupt module can only be read by a FORTRAN program in one of three ways: RBIN,RBCD, or RDP. Since the data returned by the Handler must be modified to provide the required results for RBCD and RDP, a WAITFR is included in the library routine and control is not returned to the calling program until after the results are returned.

16.6.1 RBIN: Read Binary

Read the indicated module data into the low-order 16 bits of the specified data word.

Form:	CALL RBIN (LUN,module,word[,ev])
Where:	LUN is decimal and represents the Logical Unit Number module is an integer representing the module number word is an integer variable to be set with the module data ev is the integer Event Variable
Example:	CALL RBIN (23,16,IDATA,IEV)

16.6.2 RBCD: Read BCD

Read the indicated module data as four BCD digits and set the indicated word to the corresponding value.

Form:	CALL RBCD (LUN,module,word[,ev])
Where:	LUN is decimal and represents the Logical Unit Number d. module is an integer representing the module number word is an integer variable to be set with the module data ev is the integer Event Variable
Example:	CALL RBCD (23,16,IDATA,IEV)

16.6.3 RDP: Read Point

Read the value of the indicated digital point and set a logical word to its state.

Form:	CALL RDP (LUN,point,state[,ev])
Where:	LUN is decimal and represents the Logical Unit Number point is a decimal integer and is the digital point number state is a logical variable set to the state (.TRUE. or .FALSE) of the indicated point ev is the integer Event Variable
Example:	CALL RDP (27,89,LV,IEV)

16.7 CTDI,RDDI,AND DFDI: CONTACT INTERRUPT DIGITAL INPUT -- W733 MODULES

Digital input from contact interrupt modules is reported in a requester-provided circular buffer. Each buffer entry is four words long and is of the following format:

Word	Contents
0	Entry existence indicator
1	Module number
2	COS gates output
3	Module data

The entry existence indicator is set nonzero when a buffer entry is made. When the requester has removed or processed an entry it must clear the existence indicator to free the buffer entry position. Entries are made in a circular fashion, starting at the first (low address), filling in order of increasing core addresses to the last (high address), and wrapping around from last to first. If inputs occur in a burst of sufficient duration to overrun the buffer, data are discarded and a count of data overruns is incremented. The nonzero entry existence indicator also serves as an overrun indicator. A positive value (+1) indicates no overruns between buffer entries, and a negative value is the two's complement of the number of times data was discarded between buffer entries.

The module number indicates a module on which one or more discrete points has undergone a change of state. The module number consists of the high-order bits (0-13) of point numbers.

The COS (change-of-state) gates output indicates which points on the indicated module have changed state. The position of one bit in the COS output may be used to determine the low-order bits (14-17) of point numbers, i.e., COS bits 0 and 2 represent input from points 16M+0 and 16M+2, where M is the module number.

The module data bits 0-15 indicate the polarity (state) of points 0-15 of the indicated module.

Contact interrupt input is reported to only one Task. This is controlled by two UDC Handler Task functions: CTDI, which connects a buffer for digital input, and DFDI, which disconnects a buffer from digital input. While a buffer is connected, all other connect requests are rejected, and a disconnect request is only accepted from a Task containing a buffer that is connected.

16.7.1 CTDI: Connecting a Buffer

The CTDI I/O function call connects a buffer to receive digital input from contact interrupt modules.

CTDI is only issued by a FORTRAN program in the following way:

Form:	CALL CTDI (LUN,buf,size,tev[,ev])
Where:	LUN is decimal and represents the Logical Unit Number buf is the circular buffer array size is the integer size of the buffer array tev is the Trigger Event Variable ev is the integer Event Variable
Example:	CALL CTDI (32,IBUF,ISIZE,ITEV,IEV)

The buffer must be a multiple of four words long and may be as small as four words. Whenever a buffer entry is made, the Trigger Event Variable (word 3), which is required, is set to +1, and a Significant Event is declared.

The CPB for this form of QUEUE I/O has the following form:

Word	Contents
0	7000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Trigger Event Variable address
4	First (low) address of circular buffer
5	Last (high) address of circular buffer

16.7.2 RDDI: Reading a Buffer

RDDI is a FORTRAN-callable subroutine, not an I/O function. It reads digital input from a circular buffer (connected by CTDI to the UDC) and clears the Trigger Event Variable.

The RDDI FORTRAN call is issued in the following way:

Form:	CALL RDDI (point,state[,count])[ev]
Where:	point is an integer variable which is set with the digital point number (nonzero) if a buffer entry is found state is a logical variable which is set to the state (.TRUE. or .FALSE.) of the indicated point count is set to the integer buffer overrun count (normally zero) ev is the integer Event Variable
Example:	CALL RDDI (IDP,LV,IOC)

If all digital inputs in the buffer have been reported, RDDI sets point and state to zero. If unreported digital inputs exist, they are reported by setting point and state. Since as many as sixteen digital inputs may be recorded in each buffer entry (for each trigger), RDDI is normally called repeatedly until a zero is returned as a point number.

16.7.3 DFDI: Disconnecting a Buffer

The DFDI I/O function call disconnects a digital input buffer from the UDC.

DFDI is only issued by a FORTRAN program in the following way:

Form:	CALL DFDI (LUN)[ev]
Where:	LUN is decimal and represents the Logical Unit Number ev is the integer Event Variable

Example:	Connect Task for digital input (STOP 1 if connect rejected) and request Tasks JOE, PETE, or BILL if contact closures or points 1,2, or 3 are detected, respectively; STOP 2 if input occurs on points other than the above:
	`    LOGICAL L`
	`    DIMENSION IBUF(40), TASK(3)`
	`    DATA TASK(1)/3HJOE/,TASK(2)/4HPETE/,`
	`    TASK(3)/4HBILL/`
	`C`
	`    CALL CTDI (32,IBUF,ITEV,IEV)`
	`    CALL WAITFR (IEV)`
	`    IF (IEV .LT. 0) STOP 1`
	`C`
	`100 CALL WAITFR (ITEV)`
	`200 CALL RDDI (N,L)`
	`    IF (N .EQ. 0) GO TO 100`
	`    IF (N .GT. 3) GO TO 300`
	`    IF (L) CALL REQST (TASK(N),0)`
	`    GO TO 200`
	`C`
	`300 CALL DFDI (32)`
	`    STOP 2`
	`    END`

The CPB for this form of QUEUE I/O has the following form:

Word	Contents
0	7100 (I/O function code)
1	Event Variable address
2	Logical Unit Number

16.8 SAMPLE PROGRAM

Following is a program to drive a control panel. It illustrates use of DOL, RBCD, and RDDI. The contact sense modules are wired for "pulse open" and "pulse close" operation, but the program discards contact closures (IF (LV) GO TO...).

```
C    TDSP --- TASK DISPATCHER.  TASK TO DRIVE THE SCHEDULE
C    MODULE OF THE RSX-15 DEMO PANEL.
C
C    EDIT #1             16 OCT 71
C
C    DIGITAL OUTPUT LATCHING (DOL) POINTS USED:
C         1 -- "SELECT" LAMP [READ TASK NUMBER]
C         2 -- "CANCEL" LAMP
C         3 -- "SCHED" LAMP
C         4 -- "NOW" LAMP [SYNC UNITS]
C         5 -- "SEC" LAMP
C         6 -- "MIN" LAMP
C         7 -- "HOUR" LAMP
C         8 -- "NO" LAMP [RESCHEDULING? NO/YES]
C         9 -- "YES" LAMP
C        10 -- "TICKS" LAMP [RESCHEDULE PERIOD UNITS]
C        11 -- "SEC" LAMP
C        12 -- "MIN" LAMP
C        13 -- "HOURS" LAMP
C        14 -- "EXECUTE" LAMP
C        19 -- FAULT INDICATOR LAMP
C
C    CONTACT INTERRUPT (CI) POINTS USED:
C         1 -- "SELECT" CONTACTS [READ TASK NUMBER]
C         2 -- "CANCEL" CONTACTS
C         3 -- "SCHED" CONTACTS
C         4 -- "NOW" CONTACTS [SYNC UNITS]
C         5 -- "SEC" CONTACTS
C         6 -- "MIN" CONTACTS
C         7 -- "HOUR" CONTACTS
C         8 -- "NO" CONTACTS [RESCHEDULING? NO/YES]
C         9 -- "YES" CONTACTS
C        10 -- "TICKS" CONTACTS [RESCHEDULE PERIOD UNITS]
C        11 -- "SEC" CONTACTS
C        12 -- "MIN" CONTACTS
C        13 -- "HOURS" CONTACTS
C        14 -- "EXECUTE" CONTACTS
C
C    CONTACT SENSE (CS) MODULES USED:
C         1 -- TASK NUMBER (3 BCD THUMBWHEELS)
C         2 -- RESCHEDULE PERIOD (3 BCD THUMBWHEELS)
C
        INTEGER EV,TEV,BUF
        LOGICAL LV
C
        DIMENSION BUF(40),ID(2),TASK(6),ISP(5)
C
C    TASK NAMES & NUMBERS:
        DATA TASK(1) /5HCALIB/
        DATA TASK(2) /4HCSPT/
        DATA TASK(3) /4HCSPV/
        DATA TASK(4) /3HSET/
        DATA TASK(5) /5HLTEMP/
        DATA TASK(6) /5HPTEMP/
C
        DATA ID(1),ID(2)/3,1/
        DATA ISP(2),ISP(3)/1,1/
C
        NTMAX=6
        LUN=31
C
```

```
C    CONNECT 'BUF' TO RECEIVE DIGITAL INPUT FROM CONTACT
C    INTERRUPT MODULES.  "STOP 1" IF CONNECT REJECTED.
C
           CALL CTDI (LUN,BUF,TEV,EV)
           CALL WAITFR (EV)
           IF (EV .LT. 0) STOP 1
C
C    TURN OFF ALL BUTTON LAMPS (DOL 1-14)
C
100        DO 101 J=1,14
101        CALL DOL (LUN,J,.FALSE.)
C
C    FLASH "SELECT" BUTTON (DOL #1) UNTIL IT IS PRESSED,  I.E.,
C    UNTIL A CONTACT CLOSURE IS DETECTED ON CI #1.
C
105        CALL DOL (LUN,1,.TRUE.)
           CALL MARK (ID,EV)
           CALL WAITFR (EV)
           IF (TEV .NE. 0) GO TO 110
           CALL MARK (ID,EV)
           CALL WAITFR (EV)
           IF (TEV .NE. 0) GO TO 110
           CALL DOL (LUN,1,.FALSE.)
           CALL MARK (ID,EV)
           CALL WAITFR (EV)
           IF (TEV .EQ. 0) GO TO 105
110        CALL RDDI (N,LV)
           IF (N .EQ. 0) GO TO 105
           IF (LV) GO TO 110
           IF (N .NE. 1) GO TO 110
C
C    READ REDUNDANT CI POINTS, SO THAT CONTACT BOUNCE ON
C    CI #1 IS NOT TAKEN AS A RESET QUE WHILE WAITING FOR
C    CI #2 OR CI #3.
C
           CALL MARK (ID,EV)
           CALL WAITFR (EV)
111        CALL RDDI (N,LV)
           IF (N .NE. 0) GO TO 111
C
C    A CONTACT CLOSURE HAS BEEN DETECTED ON CI #1 ("SELECT"
C    BUTTON), TURN "SELECT" BUTTON (DOL #1) OFF, TURN "FAULT"
C    BUTTON (DOL #19) OFF, AND READ TASK NUMBER 'NT' FROM
C    CONTACT SENSE MOD #1 (3 BCD THUMBWHEELS).
C
           CALL DOL (LUN,1,.FALSE.)
           CALL DOL (LUN,19,.FALSE.)
           CALL RBCD (LUN,1,NT,EV)
           CALL WAITFR (EV)
C
C    FAULT IF OUT OF RANGE TASK NUMBER
C
           IF (NT .EQ. 0) GO TO 900
           IF (NT .EQ. 999) GO TO 115
           IF (NT .GT. NTMAX) GO TO 900
C
C    TURN BOTH THE "CANCEL" & "SCHEDULE" BUTTONS (DOL #2 &
C    DOL #3) ON      AND WAIT FOR A CONTACT CLOSURE ON EITHER
C    CI #2 ("CANCEL" BUTTON) OR CI #3 ("SCHEDULE" BUTTON).
C    RESTART IF CI #1 CLOSURE.
C
115        CALL DOL (LUN,2,.TRUE.)
           CALL DOL (LUN,3,.TRUE.)
```

```
200     CALL WAITFR (TEV)
210     CALL RDDI (N,LV)
        IF (N .EQ. 0) GO TO 200
        IF (LV) GO TO 210
        IF (N .EQ. 2) GO TO 220
        IF (N .EQ. 3) GO TO 230
        IF (N .EQ. 1) GO TO 100
        GO TO 210
C
C   A CONTACT CLOSURE HAS BEEN DETECTED ON CI #2 ("CANCEL" BUTTON),
C   TURN "SCHEDULE" BUTTON (DOL #3) OFF, TURN "GO" BUTTON
C   (DOL #14) ON, AND WAIT FOR A CONTACT CLOSURE ON CI #14
C   ("GO" BUTTON).  RESTART IF CI #1 CLOSURE.
C
220     CALL DOL (LUN,3,.FALSE.)
        CALL DOL (LUN,14,.TRUE.)
221     CALL WAITFR (TEV)
222     CALL RDDI (N,LV)
        IF (N .EQ. 0) GO TO 221
        IF (LV) GO TO 222
        IF (N .EQ. 1) GO TO 100
        IF (N .NE. 14) GO TO 222
C
C   A CONTACT CLOSURE HAS BEEN DETECTED ON CI #14 ("GO" BUTTON).
C   TURN "GO" BUTTON OFF (DOL #14) AND CANCEL INDICATED
C   TASK.  SPECIAL CASE IF CANCEL TASK #999: EXIT TASK
C   DISPATCHER.
C
        IF (NT .EQ. 999) GO TO 999
        CALL CANCEL (TASK(NT),EV)
        IF (EV .LT. 0) GO TO 900
        GO TO 100
C
C   A CONTACT CLOSURE HAS BEEN DETECTED ON CI #3 ("SCHEDULE"
C   BUTTON). TURN "CANCEL" BUTTON OFF (DOL #2), TURN SCHEDULE
C   TIME BUTTONS ("NOW", DOL #4; "NEXT SEC", DOL #5; "NEXT MIN",
C   DOL #6; & "NEXT HR", DOL #7) ON, AND WAIT FOR A CONTACT
C   CLOSURE ON EITHER OF THE FOUR BUTTONS (CONTACT INTERRUPT
C   POINTS 4-7).  RESTART IF CI #1 CLOSURE.
C
230     CALL DOL (LUN,2,.FALSE.)
        IF (NT .EQ. 999) GO TO 900
        DO 231 J=4,7
231     CALL DOL (LUN,J,.TRUE.)
232     CALL WAITFR (TEV)
233     CALL RDDI (N,LV)
        IF (N .EQ. 0) GO TO 232
        IF (LV) GO TO 233
        IF (N .EQ. 1) GO TO 100
        IF (N .LT. 4) GO TO 233
        IF (N .GT. 7) GO TO 233
C
C   A CONTACT CLOSURE HAS BEEN DETECTED ON CI #N (WHERE
C   3<N<8). RECORD SCHEDULE TIME (SYNC UNITS 'ISP(1)'), AND
C   TURN OFF THREE BUTTONS NOT PRESSED.  'ISP(1)' -- 1, NOW;
C   2, NEXT SEC; 3, NEXT MIN; & 4, NEXT HR.
C
        ISP(1)=N-3
        DO 235 J=4,7
        IF (J .EQ. N) GO TO 235
        CALL DOL (LUN,J,.FALSE.)
235     CONTINUE
C
```

```
C    TURN ON RESCHEDULE "NO" & "YES" BUTTONS (DOL #8 & DOL #9),
C    AND WAIT FOR A CONTACT CLOSURE ON EITHER BUTTON (CI #8 OR
C    CI #9).  RESTART IF CI #1 CLOSURE.
C
         CALL DOL (LUN,8,.TRUE.)
         CALL DOL (LUN,9,.TRUE.)
310      CALL WAITFR (TEV)
320      CALL RDDI (N,LV)
         IF (N .EQ. 0) GO TO 310
         IF (LV) GO TO 320
         IF (N .EQ. 8) GO TO 380
         IF (N .EQ. 9) GO TO 390
         IF (N .EQ. 1) GO TO 100
         GO TO 320
C
C    A CONTACT CLOSURE HAS BEEN DETECTED ON CI #8 ("NO" BUTTON),
C    TURN OFF "YES" BUTTON (DOL #9) AND ZERO RESCHEDULE DELTA
C    [ISP(4)] & RESCHEDULE UNITS [ISP(5)], TURN "GO" BUTTON (DOL #14)
C    ON, AND WAIT FOR CONTACT CLOSURE ON CI #14 (GO BUTTON).
C
380      CALL DOL (LUN,9,.FALSE.)
         ISP(4)=0
         ISP(5)=0
         GO TO 600
C
C    A CONTACT CLOSURE HAS BEEN DETECTED ON PT #9 ("YES" BUTTON),
C    TURN OFF "NO" BUTTON, AND PROCEDE TO DETERMINE RESCHEDULE
C    DELTA [ISP(4)] & RESCHEDULE UNITS [ISP(5)].
C
390      CALL DOL (LUN,8,.FALSE.)
C    TURN ON RESCHEDULE UNITS BUTTONS ("TICK", DOL #10; "SEC",
C    DOL #11; "MIN", DOL #12; "HOUR", DOL #13), AND WAIT
C    FOR A CONTACT CLOSURE ON ONE OF THE BUTTONS (CONTACT
C    INTERRUPT POINTS 10-13).  RESTART IF CI #1 CLOSURE.
C
         DO 410 J=10,13
410      CALL DOL (LUN,J,.TRUE.)
411      CALL WAITFR (TEV)
412      CALL RDDI (N,LV)
         IF (N .EQ. 0) GO TO 411
         IF (LV) GO TO 412
         IF (N .EQ. 1) GO TO 100
         IF (N .LT. 10) GO TO 412
         IF (N .GT. 13) GO TO 412
C
C    A CONTACT CLOSURE HAS BEEN DETECTED ON CI #N (WHERE
C    9<N<14), RECORD RESCHEDULE UNITS [ISP(5)], AND TURN
C    OFF THREE BUTTONS NOT PRESSED.  'ISP(5)' -- 1, TICK;
C    2, SEC; 3, MIN; & 4, HR.
C
         ISP(5)=N-9
         DO 420 J=10,13
         IF (J .EQ. N) GO TO 420
         CALL DOL (LUN,J,.FALSE.)
420      CONTINUE
C
C    READ RESCHEDULE DELTA FROM CONTACT SENSE MOD #2 (3 BCD
C    THUMBWHEELS).
C
         CALL RBCD (LUN,2,ISP(4),EV)
         CALL WAITFR (EV)
C
C    LIMIT PERIOD OF CYCLIC RESCHEDULING TO ONE DAY.  SINCE
```

```
C    ONLY THREE DIGITS MAY BE INPUT, THE ONLY CHANCE OF
C    EXCEEDING 24 HRS IS WHEN THE UNITS ARE HOURS (4).
C
          IF (ISP(5) .NE. 4) GO TO 600
          IF (ISP(4) .GT. 24) GO TO 900
C
C    A SCHEDULE HAS BEEN ESTABLISHED (ONE TIME OR CYCLIC).
C    TURN "GO" BUTTON (DOL #14) ON, AND WAIT FOR A CONTACT
C    CLOSURE ON CI #14 ("GO" BUTTON).  RESTART IF CI #1
C    CLOSURE.
C
600       CALL DOL (LUN,14,.TRUE.)
610       CALL WAITFR (TEV)
620       CALL RDDI (N,LV)
          IF (N .EQ. 0) GO TO 610
          IF (LV) GO TO 620
          IF (N .EQ. 1) GO TO 100
          IF (N .NE. 14) GO TO 620
C
C    A CLOSURE HAS BEEN DETECTED ON CI #14 ("GO" BUTTON),
C    SYNC DIRECTIVE AND  RESTART DISPATCH CYCLE.
C
          CALL SYNC (TASK(NT),ISP,0,EV)
          IF (EV .LT. 0) GO TO 900
          GO TO 100
C
C    A FAULT HAS OCCURRED. SET "FAULT" LITE AND RESTART
C
900       CALL DOL (LUN,19,.TRUE.)
          GO TO 100
C
C    EXIT TASK DISPATCHER ("TASK" #999 CANCELLED)
C
999       CALL DOL (LUN,2,.FALSE.)
          CALL DOL (LUN,14,.FALSE.)
          CALL DFDI (LUN)
          STOP
C
          END
```

CHAPTER 17

COMMON COMMUNICATOR I/O

The COMMON Communicator is the intermediary between System COMMON Blocks and USER-mode tasks. Because USER-mode tasks run with memory relocation and protection, they cannot directly access System COMMON Blocks, which are external to the task partitions, unless they use the XVM/RSX core sharing feature. EXEC-mode tasks can also (but need not) use this task.

17.1 CC HANDLER TASK

The COMMON Communicator, associated with device name CC, is structured like an I/O device Handler called CC.... and handles communication via core-to-core block transfers directed by special forms of the QUEUE I/O Directive. Up to 32K of System COMMON Blocks can be accessed per QUEUE I/O request.

17.2 CC HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the CC Handler Task. Following are legal function calls which may be issued for CC service and a brief description of each:

Table 17-1
COMMON Communicator I/O Functions

CC Call	Function
HINF	Request Handler Task information
ATTACH	Obtain exclusive use of COMMON Communicator
DETACH	Release attached COMMON Communicator
COMGET	Transfer data from COMMON to Task
COMPUT	Transfer data from Task to COMMON

They are basic I/O calls and operate almost exactly as described in Chapter 2. When requested by the CC Handler Task, however, HINF does require the few special considerations discussed below.

17.2.1 HINF: Requesting Handler Task information

If a request for HINF is accepted, the Event Variable is set to +300014 (octal) to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 3 to indicate input and output device
3	Set 0 to indicate non-directory-oriented Handler
4-11	Device unit 0
12-17	Device code 14 (CC)

17.3 COMGET: TRANSFERRING DATA FROM A COMMON BLOCK TO A TASK

The COMGET FORTRAN-callable subroutine transfers data from a System COMMON Block to a user Task. It is in a library program called COMCOM.

COMGET is issued by a FORTRAN program in the following way:

Form:	CALL COMGET (LUN,wc,vn,scn[,offset][,ev])
Where:	LUN is decimal and represents the Logical Unit Number wc is the positive decimal integer word count vn is the variable name in the issuing Task to which the contents of scn are transferred scn is the System COMMON name, a string of one to five ASCII characters offset is the positive decimal integer offset (default is zero) ev is the integer Event Variable
Example:	See COMPUT example

The CPB follows:

Word	Contents
0	3000 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table has the following form:

Word	Contents
0	.SIXBT system COMMON Block name (first half)
1	.SIXBT system COMMON Block name (second half)
2	Offset from base of COMMON Block
3	Current address in Task area
4	Decimal word count (number of words to transfer)

The information from the Control Table is not queued along with the information from the CAL Parameter Block. Therefore, after making the I/O request, one must not modify the Control Table until after completion of the request.

17.4 COMPUT: TRANSFERRING DATA FROM A TASK TO A COMMON BLOCK

The COMPUT FORTRAN-callable subroutine transfers data from a user Task to a System COMMON Block. It is in a library program called COMCOM.

COMPUT is issued by a FORTRAN program in the following way:

Form:	CALL COMPUT (LUN,wc,vn,scn[,offset][,ev])
Where:	LUN is decimal and represents the Logical Unit Number wc is the positive decimal integer word count vn is the variable name in the issuing Task whose contents are transferred to scn scn is the System COMMON name, a string of one to five ASCII characters offset is the positive decimal integer offset (default is zero) ev is the integer Event Variable
Example:	Test COMGET and COMPUT:

```
C       EDIT 1  9/24/71
C       TESTS F4 CALLABLE COMGET AND COMPUT.
C
        DIMENSION KDIM(10),RDIM(10),KDSCR(10),RDSCR(10)
        COMMON KCOM(10),RCOM(10),KCSCR(10),RCSCR(10)
C
C       READ IN PARAMETERS VIA LUN 7.
C       MESSAGE OUTPUT VIA LUN 4.
C       COMMON COMMUNICATION VIA LUN 6.
C
2       WRITE(4,6)
6       FORMAT('  TYPE LUN WC OFFSET EXITSW')
        READ(7,)LUN,IWC,IOFF,IEXIT
C
C       IF NON-ZERO IEXIT READ IN, EXIT FROM TASK.
C
        IF(IEXIT.NE.0)CALL EXIT
C
C       INIT ARRAYS.
C
        DO 1 I=1,10
        KDSCR(I)=0
        KCSCR(I)=0
        RDSCR(I)=0.
        RCSCR(I)=0.
        KDIM(I)=I
        KCOM(I)=-I
        RDIM(I)=I
1       RCOM(I)=-I
```

```
C
C       PUT KDIM AND KCOM ARRAYS IN SYSTEM COMMON.
C
        CALL COMPUT(LUN,IWC,KDIM(1),'ABCDE',IOFF,IEV)
        CALL WAITFR(IEV)
        IF(IEV.NE.1)WRITE(4,)IEV
        CALL COMPUT(LUN,IWC,KCOM(1),'WXYZ',IOFF,IEV)
        CALL WAITFR(IEV)
        IF(IEV.NE.1)WRITE(4,)IEV
C
C       GET SYSTEM COMMON INTO KDSCR AND KCSCR ARRAYS.
C
        CALL COMGET(LUN,IWC,KDSCR(1),'WXYZ',IOFF,IEV)
        CALL WAITFR(IEV)
        IF(IEV.NE.1)WRITE(4,)IEV
        CALL COMGET(LUN,IWC,KCSCR(1),'ABCDE',IOFF,IEV)
        CALL WAITFR(IEV)
        IF(IEV.NE.1)WRITE(4,)IEV
C
C       REPEAT PROCESS FOR REAL ARRAYS RDIM AND RCOM.
C       DON'T SPECIFY AN EVENT VARIABLE.
        IWC=2*IWC
        IOFF=2*IOFF
C
        CALL COMPUT(LUN,IWC,RDIM(1),'ABCDE',IOFF)
        CALL COMPUT(LUN,IWC,RCOM(1),'WXYZ',IOFF)
        CALL COMGET(LUN,IWC,RDSCR(1),'WXYZ',IOFF)
        CALL COMGET(LUN,IWC,RCSCR(1)'ABCDE',IOFF)
C
C       CHECK DATA.  SCRATCH ARRAYS SHOULD BE NEGATIVE OF
C       ORIGINAL ARRAYS.
C
        DO 3 I=1,10
        IF(KDIM(I)+KDSCR(I).NE.0)WRITE(4,)KDIM(I),KDSCR(I),I
        IF(KCOM(I)+KCSCR(I).NE.0)WRITE(4,)KCOM(I),KDSCR(I),I
        IF(RDIM(I)+RDSCR(I).NE.0.)WRITE(4,)RDIM(I),RDSCR(I),I
        IF(RCOM(I)+RCSCR(I).NE.0.)WRITE(4,)RCOM(I),RCSCR(I),I
3       CONTINUE
C
C       TEST DONE.
C
        WRITE(4,5)
5       FORMAT(1X,'TEST DONE')
        GO TO 2
        END
```

The CPB follows:

Word	Contents
0	3100 (I/O function code)
1	Event Variable address
2	Logical Unit Number
3	Control Table address

The Control Table has the following form:

Word	Contents
0	.SIXBT system COMMON Block name (first half)
1	.SIXBT System COMMON Block name (second half)
2	Offset from base of COMMON Block
3	Current address in Task area
4	Decimal word count (number of words to transfer)

The information from the Control Table is not queued along with the information from the CAL Parameter Block. Therefore, after making the I/O request, one must not modify the Control Table until after completion of the request.

CHAPTER 18

XY PLOTTER I/O

18.1 XY HANDLER TASK

The I/O Device Handler Task responsible for servicing requests for the XY plotter is named XY.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is XY for purposes of reassignment. The XY plotter is interfaced to a PDP-11, so a UNICHANNEL configuration is required for this handler. Therefore, the handler task must reside in the XVM so that it is within the PDP-11 addressing space. That is, for an 8K PDP-11, it must be below 50000 and for a 12K PDP-11 it must be below 40000. The handler task requires 1400 for a partition size.

The Device Handler can be assembled to support two PDP-11 interfaces, the XY11 and the XY311. The XY11 supports a single-pen 300-step-per-second plotter; the XY311 supports a triple-pen 1800-step-per-second plotter.

18.2 XY HANDLER REQUESTS

The following table lists the legal QUEUE I/O Directives that can be made to XY....:

Table 18-1
XY Plotter I/O Functions

Call	Functions
ABORT	Abort I/O for a task
ATTACH	Obtain exclusive use of the plotter
CLOSE	Force output of any buffered lines
DETACH	Release ATTACHed plotter
ENTER	Reset character size to default
HINF	Obtain Handler Task information
READ	Obtain current state of the plotter
WRITE	Output lines, characters etc. to plotter

18.2.1 ABORT

This function is entirely standard; see Chapter 2.

18.2.2 ATTACH

This function is entirely standard; see Chapter 2.

18.2.3 CLOSE

There are two levels of buffering for plotter information. The handler collects 10 XY pairs for line commands for each request to the PDP-11, to minimize interprocessor overhead. The PDP-11 SPOOLER, if activated, buffer plotter requests onto the disk. The CLOSE Directive forces any lingering information in either buffer to go out to the PDP-11. It is recommended to issue a CLOSE at the end of all plotter programs.

18.2.4 DETACH

This functon is entirely standard; see Chapter 2.

18.2.5 ENTER

The PDP-11 keeps information as to the size and shape of plotted characters. The ENTER directive resets these parameters to their default values. The standard default is to plot characters along the +X Axis with a size of 1/5" by 1/5".

18.2.6 HINF

If a HINF request is accepted, the Event Variable is set to 100026. The 26 is the device code for the plotter. The leading 1 says that this is an output-only device. (The READ directive returns information to the user, but does no input I/O.)

18.2.7 READ

The READ directive obtains plotter status from the handler, but does no physical I/O. The returned buffer contains 8 words for a XY11, and 9 for the XY311, as described in the following table:

Table 18-2
Format of Information Returned by READ

Word#	Contents (octal integers)
1	400000, control word for FORTRAN, ignore
2	present pen position, X Co-ordinate
3	present pen position, Y Co-ordinate
4	X size of character, in steps
5	Y size of character, in steps
6	200000*SIN(Z), where Z is the angle of the character
7	200000*COS(Z). string from horizontal.
8	0 if pen up, 100000 if down
9	if XY311, which pen in use (1-3)

18.2.8 WRITE

The WRITE Directive is used to output character information, line information and control information to the plotter. Co-ordinate data are in integer plotter steps. Scaling is the responsibility of the user. The XY11 has 100 steps per inch; the XY311 500 steps per inch. Plus X is toward the paper feed roll.

IOPS ASCII is supported in the standard format with the following exceptions. The control characters, those in the octal range 0-37, are totally ignored, not even being sent to the PDP-11. Characters in the range 140-177 are sent as characters in the range of 100-137, thus converting lower case to upper case. The character count of the I/O directive, rather than control characters, is used to terminate the string. Only the first 132 characters are sent across to the PDP-11, regardless of the size of the arriving string.

Each character is plotted inside a rectangular boundary of lX by lY plotter steps. The size of these values are specified under mode 10 of IOPS BINARY write. The X direction spacing includes the intercharacter spacing so that the first and second characters share a vertical boundary.

IOPS BINARY is used for all of the remaining type of plotter control. The first data word of the buffer is an integer mode word whose value ranges from 0-10 decimal (11 for the XY311). There follow in the buffer a variable number of integer arguments, as described in the following table:

Table 18-3
IOPS Binary Modes

Mode No	Arguments	Action
0	None	Lift pen
1	None	Lower pen
2	n,IX,IY pairs	Lift pen, move to absolute co-ordinates specified by each IX,IY pair
3	n,IX,IY pairs	Lower pen, move to absolute co-ordinates specified by each IX,IY pair
4	n,IX,IY pairs	Lift pen, move delta x, delta y specified by each IX,IY pair
5	nIX,IY pairs	Lower pen, move delta x, delta y specified by each IX,IY pair
6	count,char. string	Alternate character method; a char. count, followed by the string
7	IX,IY	Define present position as co-ordinates IX,IY
8	nIX,IY pairs	Leave pen, move to the absolute co-ordinates specified by each IX,IY pair
9	nIX,IY pairs	Leave pen, move delta x, delta y specified by each IX,IY pair
10	IX,IY,ISIN,ICOS	Set character size to IX wide, IY high; character string angle Z gives 200000*SIN(Z) for ISIN, and 200000*COS(Z) for ICOS.
11	number	XY311 select pen (1-3)

18.3 UNICHANNEL OPERATION

The presence of the PDP-11 is largely transparent to the plotter user, but not entirely. The plotter interface basically doesn't recognize errors, for example, out of paper, out of ink, pen pinned on edge of plotter. There is not even an off line switch, to allow clean operator intervention, on the XY11. A software switch was built into the PDP-11 PIREX monitor. If console switch #2 is raised, PIREX acts as if the plotter is offline. No notification comes back to the XVM; the plotter resumes when the switch is again lowered. The XY311 does have an offline switch, and switch #2 is no longer operational (PIREX has an assembly parameter for plotter type). For the XY211 there is still no message at the XVM.

If the PIREX SPOOLING facility is turned on under DOS, and RSX is then invoked, SPOOLING remains active under RSX. This means that a plot job under RSX generates a plot file in the SPOOLER, and may then complete long before the physical plot is done. Additional jobs may then ATTACH to the plotter and generate additional plotter output to be plotted later. However, it is the responsibility of these plotter jobs to ensure that they do not use the same space on the plotter. The SPOOLER will continue from one job to the next, not allowing time for operator intervention. A job separation standard should be adopted suitable for each installation. The handler could not make an arbitrary spacing assumption; applications have already been observed that create "layered" plots from several inputs.

18.4 FORTRAN EXAMPLES

The following program fragments show the use of FORTRAN to generate calls to the XY Plotter Handler Task. It is assumed that the XY plotter has been reassigned to LUN 7.

18.4.1 Use of IOPS ASCII

```
          S=2.
          SQ=SQRT(S)
          WRITE (7,100) SQ
100       FORMAT (1X, 5HSQ=,E15.8)
```

18.4.2 Characters From IOPS Binary

```
          DIMENSION   T(4)
          DATA  T(1)/5H  ABCD/,T(2)/5HEFGHI/
          IM=6
          IC=9
          WRITE  (7)  IM, IC, T
```

(Note: The array may also be filled by I/O.)

18.4.3 Make Characters Double Sized and Vertical (XY11)

```
IM=10
IX=40
IY=40
IS=#200000
IC=0
WRITE  (7) IM, IX, IY, IS, IC
```

18.4.4 Transfer an Array of 25 Lines

```
DIMENSION  IL(50)
IM=3
WRITE  (7) IM, IL
```

18.4.5 Recommended Close-out for Plotter Job (LUN=7)

```
CALL CLOSE (7,5H@@@@@,3H@@@,IEV)
CALL WAITFR (IEV)
```

CHAPTER 19

CARD PUNCH INPUT/OUTPUT

19.1 CARD PUNCH HANDLER TASK

The Input/Output Device Handler Task responsible for servicing requests for the card punch is called CP.... and handles calls submitted through special forms of the QUEUE I/O Directive. The device name is CP for purposes of reassignment.

19.2 CP HANDLER REQUESTS

Special forms of the QUEUE I/O Directive can be issued to queue requests to the CP I/O Device Handler Task. Following are legal function calls which may be issued for CP service and a brief description of each:

Table 19-1
Card Punch I/O Functions

CP Call	Function
HINF	Request Handler Task information
ATTACH	Obtain Exclusive use of card punch
DISCONNECT & EXIT	Disconnect Handler from interrupt line and exit
DETACH	Release attached card punch
WRITE	Punch output from buffer
CLOSE	Close punch file
ABORT	Abort I/O for Task

19.2.1 HINF: Requesting Handler Task Information

If a request for HINF is accepted, the Event Variable is set to +100022 (octal) to indicate the following:

Bit	Contents
0	Set 0 to make Event Variable positive
1-2	Set 1 to indicate an output-only device

Bit	Contents
3	Set 0 to indicate a non-directory-oriented Handler
4-11	Device unit 0
12-17	Device code 22 (card punch)

19.3 WRITE: WRITING OUTPUT FROM A BUFFER

The WRITE I/O function call, WRITEs output from a buffer to the CP Handler Task assigned to the specified Logical Unit Number. It operates almost exactly as in the description of WRITE in Chapter 2, with the following exceptions and special features.

If the punch is not ready, the following message is output to Logical Unit 3 (the MCR terminal):

```
*** CP NOT READY
```

The card punch handler then waits until the device is ready. Unlike some other I/O Device Handler Tasks, the CP Handler Task will not SUSPEND under such circumstances. Hence, it is never required that the operator RESUME this Handler Task.

The only legal data mode for WRITEs to this Handler Task is IOPS ASCII (mode 2). In this mode up to 80 characters can be punched per card. Prior to punching each ASCII character, the CP Handler Task converts the ASCII code into either of two Hollerith codes (DEC026 or DEC029). The codes punched are compatible with those expected by the Card Reader I/O Handler Task.

19.4 CLOSE: CLOSING A FILE

The CLOSE I/O function call informs the CP Handler Task assigned to the specified Logical Unit Number that the issuing task has completed a set of related I/O functions on the current file. The CLOSE directive results in the punching of an End-of-File card.

APPENDIX A

EVENT VARIABLES

Event Variables are software flags set by RSX for system or other Tasks. When issuing an I/O function from a MACRO or FORTRAN Task, the user can specify an Event Variable in the I/O call itself. It is set to inform him about the status of the I/O request and the success of the I/O operation. In all but a few cases, inclusion of this variable is optional.

If it is included, it usually takes the form of a variable to which has been assigned the address to contain the Event Variable. Word 1 of the CPB normally contains the address. If no Event Variable has been specified, the contents of this word remains zero. If an Event Variable has been specified, the contents of the address pointed to by Word 1 are initially set to zero to indicate that the function has not yet been processed.

If the request cannot be queued, -101,-102,-103, or -777 (see Table A-1) is returned. If the request is successfully queued, the Event Variable remains zero, indicating that the request is pending. Eventually, when the requested operation is attempted, the Event Variable is set either to a positive value, indicating success, or to a negative number whose value pinpoints the reason for failure of the operation. Thus the returned code is in the following ranges:

Code	Meaning
+n	Function succeeded; n is almost always 1
0	Function pending
-n	Function failed; n is a number indicating why failure occurred

A Significant Event is declared whenever an Event Variable is set.

Some Event Variables are specific to a particular function; others are common to two or more functions. Table A-1 provides a summary of all negative Event Variables that may be returned to RSX I/O functions, as well as possible reasons for failure. Table A-2 lists special meanings some Event Variables may have in reference to certain functions.

Table A-1
Returned Event Variables

Event Variable	Reason
1 +1	Successful completion
777 -1	No floating-point hardware on this machine
773 -5	Illegal header word read from device; data mode incorrect or data validity bits improperly set
772 -6	Unimplemented or illegal I/O function
771 -7	Illegal I/O data mode
770 -10	Some file still open by same Task on same LUN
767 -11	File not open
766 -12	DECtape or Magtape error or illegal block pointer
765 -13	File not found
764 -14	Directory full
763 -15	I/O device medium full
7[illegible] -16	Output word-pair-count or input buffer-size error
761 -17	Read/compare error
760 -20	Backspace illegal at beginning of tape
757 -21	End-of-tape reached
755 -23	Input word-pair-count error
754 -24	LUN has been reassigned while request was in I/O request queue
753 -25	Buffer preallocation unsuccessful; insufficient room remains in partition
752 -26	Illegal function for a USER-mode Task
751 -27	Nonexistent disk unit number
750 -30	Address for USER-mode Task is outside the Task's partition
747 -31	I/O parameter exceeds COMMON Block bounds
746 -32	Nonexistent System COMMON Block

(Continued on next page)

Table A-1 (Cont.)
Returned Event Variables

Event Variable	Reason
745 -33	Unit dismounted or directory not initialized
744 -34	Data missed
743 -35	Illegal device code or nonexistent disk type
742 -36	Nonexistent block number
730 -50	Rename error; file already exists in the Directory
727 -51	Illegal to read a truncated file
726 -52	Input file has no end of file
725 -53	Illegal (null) file name or extension
724 -54	This file already open
723 -55	This file already open for modification
722 -56	No I/O buffer available
721 -57	Sequential/random-access file name conflict
720 -60	Random-access file size error
717 -61	Random-access file accounting information error
716 -62	Random-access file size error; size requested is negative or greater than 777 octal blocks
710 -70	I/O parameter error
707 -71	Buffer connect or disconnect error
706 -72	Stop I/O or start I/O error
705 -73	Add-to-memory overflow
701 -77	Violation of restricted usage of Directive
677 -101	LUN out of range
676 -102	LUN not assigned to a physical device
675 -103	Nonresident or noninitialized I/O Device Handler Task or RASP not running

(Continued on next page)

Table A-1 (Cont.)
Returned Event Variables

Event Variable	Reason
-104	Parameter (in CPB or Control Table)
-201	Task not in system
-202	Task is active Task is inactive
-203	Request not Task-issued
-204	Task is disabled
-205	Task not suspended
-206	Illegal Task priority
-207	Task already fixed Task not fixed
-210	Partition occupied
-211	Partition not in system
-212	Partition for Task's STL node lost because of reconfiguration
-213	Partition assigned to Task currently being reconfigured
-301	Line number rejected
-302	Line is connected Line is not connected
-777	Deque node unavailable (empty pool)
nnn	I/O hardware error, where nnn is the device status word; because the error flag is represented by bit 0, nnn is negative

Table A-2
Special Meanings of Event Variables

Event Variable	Function	Reason
+n	HINF	Special code of device and Handler information
+4	WRAOF, FSPREC, FSPFIL, FSPEOT, MTGET, MTPUT, READ, WRITE	Successful completion; end-of-tape encountered; for Magtape, +4 overrides any other setting
+3	MTGET, MTPUT, READ	Successful completion; end-of-file encountered
+3	MTGET, FSPFIL	Successful completion; end-of-file mark encountered; forward spacing terminated and tape head left positioned following file mark
	BSPREC, BSPFIL	Successful completion; end-of-file mark encountered; backspacing terminated and tape head left positioned before file mark
	WREOF	Successful completion; end-of-file written; head positioned after end-of-file mark
	FSPEOT	Successful completion; forward spacing terminated and tape head left positioned after logical end-of-tape mark
+2	READ requests to Terminal Handler	Carriage return terminated line (IOPS ASCII)
	WRITE requests to Terminal Handler	CTRL/U aborted output (IOPS ASCII)
+1	READ	Successful completion, but parity, checksum, or buffer overflow errors may have occurred
+1	READ requests to Terminal Handler	ALTMODE terminated line (IOPS ASCII)

(continued on next page)

Table A-2 (Cont.)
Special Meanings of Event Variables

Event Variable	Function	Reason
	ATTACH	Successful completion or ATTACH request is redundant and is ignored
	DETACH	Successful completion or DETACH request is redundant and is ignored
-5	READ requests to disk file Handlers	Illegal header word, header data mode does not match READ data mode; end-of-file indicator is an exception because mode is 5
-15	Requests to Disk Driver	Insufficient contiguous free storage available on any single disk platter
-16	Requests to Terminal Handler	IOPS ASCII input -- line buffer size less than 4; IMAGE ASCII input -- line buffer size less than 3; IMAGE ASCII output -- line buffer header word-pair-count less than 2
	Request to Paper Tape Punch Handler	Word-pair-count in the line buffer header less than 2
	Requests to Paper Tape Reader Handler	Illegal buffer size (from CPB); size less than 3 for data modes 0, 1, and 3; size less than 4 for data mode 2
	Requests to disk file Handlers	Illegal input buffer size (less than 3) or illegal output word-pair-count (zero or greater than 200 octal)
-21	Requests to Magtape Handler	Event Variable was +4 and an attempt to move tape forward has been rejected
-23	Requests to Paper Tape Reader Handler	Word-pair-count is 0 in header word 0 , which was read from the tape in IOPS BINARY mode
-73	ADSSET, ADRSET	Add-to-memory overflow; for FORTRAN convenience, when this error occurs, the Memory Overflow Event Variable (if included) will be set to 1 plus the number of the overflowing channel

(Continued on next page)

Table A-2 (Cont.)
Special Meanings of Event Variables

Event Variable	Function	Reason
-104	Requests to Disk Driver	Error in Control Table argument; may signify: 1. Amount of storage to be allocated is negative, zero, or greater than 377400 octal 2. Amount of storage to be deallocated is negative, zero, or not a multiple of 256 (400 octal) 3. Illegal disk platter number; number is negative or too large 4. Disk address is not a multiple of 356 (400 octal) 5. Disk address plus the amount of storage to be allocated or deallocated indicates ALLOCATE or DEALLOCATE request is attempting to deal with more than 511 contiguous blocks
-n	Requests to Magtape Handler	Magtape error status when user recovery is specified
	Requests to Disk Driver	Persistent disk error; -n represents the contents of the disk status register

APPENDIX B

THE MACRO DEFINITIONS FILE

All system MACROs implemented in the RSX system are defined in one of the files supplied as part of RSX and known as the MACRO Definitions File. This Appendix contains definitions only for currently implemented standard system MACROs used for I/O. The manual on System Directives describes all others.

B.1 READ: READING FROM AN I/O DEVICE

```
.DEFIN  READ,LUN,MODE,BUFF,SIZE,EV
CAL     .+2
JMP     .+7
2600
EV+0
.DEC; LUN; .OCT
MODE
BUFF
SIZE
.ENDM
```

B.2 WRITE: WRITING TO AN I/O DEVICE

```
.DEFIN  WRITE,LUN,MODE,BUFF,EV
CAL     .+2
JMP     .+6
2700
EV+0
.DEC; LUN; .OCT
MODE
BUFF
.ENDM
```

B.3 DSKAL: RESERVING STORAGE

```
.DEFIN  DSKAL,CTB,EV
CAL     .+2
JMP     .+5
1500
EV+0
1
CTB
.ENDM
```

B.4 DSKDAL: FREEING STORAGE

```
.DEFIN  DSKDAT,CTB,EV
CAL     .+2
JMP     .+5
1600
EV+0
1
CTB
.ENDM
```

B.5 DSKPUT: WRITING TO DISK

```
.DEFIN  DSKPUT,CTB,EV
CAL     .+2
JMP     .+5
3100
EV+0
1
CTB
.ENDM
```

B.6 DSKGET: READING FROM DISK

```
.DEFIN  DSKGET,CTB,EV
CAL     .+2
JMP     .+5
3000
EV+0
1
CTB
.ENDM
```

B.7 MDALLO: RESERVING STORAGE -- MULTIPLE DISKS

```
.DEFIN  MDALLO,CTB,UNIT,TYPE,EV
CLL
LAC     (UNIT
ALS     17
DAC     CTB+1
CAL     .+2
JMP     .+6
11500
EV
1
CTB
TYPE
.ENDM
```

B.8 MDDEAL: FREEING STORAGE -- MULTIPLE DISKS

```
.DEFIN  MDDEAL,CTB,UNIT,TYPE,EV
CLL
LAC     CTB+1
AND     (7777
DAC     CTB+1
LAC     (UNIT
ALS     17
TAD     CTB+1
DAC     CTB+1
CAL     .+2
JMP     .+6
11600
EV
1
CTB
TYPE
.ENDM
```

B.9 MDGET: READING FROM MULTIPLE DISKS

```
.DEFIN  MDGET,CTB,UNIT,TYPE,EV
CLL
LAC     CTB
AND     (7777
DAC     CTB
LAC     (UNIT
ALS     17
TAD     CTB
DAC     CTB
CAL     .+2
JMP     .+6
13000
EV
1
CTB
TYPE
.ENDM
```

B.10 MDPUT: WRITING TO MULTIPLE DISKS

```
CLL
LAC     CTB
AND     (7777
DAC     CTB
LAC     (UNIT
ALS     17
TAD     CTB
DAC     CTB
CAL     .+2
JMP     .+6
13100
EV
1
```

```
        CTB
        TYPE
        .ENDM
```

B.11 ATTACH: ATTACHING AN I/O DEVICE

```
        .DEFIN  ATTACH,LUN,EV
        CAL     .+2
        JMP     .+4
        2400
        EV+0
        .DEC, LUN; .OCT
        .ENDM
```

B.12 DETACH: FREEING AN ATTACHED I/O DEVICE

```
        .DEFIN  DETACH,LUN,EV
        CAL     .+2
        JMP     .+4
        2500
        EV+0
        .DEC; LUN; .OCT
        .ENDM
```

B.13 SEEK: OPENING A FILE FOR INPUT

```
        .DEFIN  SEEK,LUN,FLNAM,EXT,EV
        CAL     .+2
        JMP     .+7
        3200
        EV+0
        .DEC; LUN; .OCT
..=.;   .SIXBT "FLNAM"
        0; .LOC ..+2
        .SIXBT "EXT"
        .ENDM
```

B.14 ENTER: OPENING A FILE FOR OUTPUT

```
        .DEFIN  ENTER,LUN,FLNAM,EXT,EV
        CAL     .+2
        JMP     .+7
        3300
        EV+0
        .DEC; LUN; .OCT
..=.;   .SIXBT "FLNAM"
        0; .LOC ..+2
        .SIXBT "EXT"
        .ENDM
```

B.15 DELETE: DELETING A FILE

```
          .DEFIN  DELETE,LUN,FLNAM,EXT,EV
          CAL     .+2
          JMP     .+7
          3500
          EV+0
          .DEC; LUN; .OCT
   ..=.;  .SIXBT "FLNAM"
          0; .LOC ..+2
          .SIXBT "EXT"
          .ENDM
```

B.16 CLOSE: CLOSING A FILE

```
          .DEFIN  CLOSE,LUN,FLNAM,EXT,EV
          CAL     .+2
          JMP     .+7
          3400
          EV+0
          .DEC; LUN; .OCT
   ..=.;  .SIXBT  'FLNAM'
          0; .LOC ..+2
          .SIXBT  'EXT'
          .ENDM
```

B.17 RENAME: RENAMING A HLE

```
          .DEFIN  RENAME,LUN,FLNAM,EXT,EV
          CAL     .+2
          JMP     .+7
          3700
          EV+0
          .DEC; LUN; .OCT
   ..=.;  .SIXBT "FLNAM"
          0; .LOC ..+2
          .SIXBT  "EXT"
          .ENDM
```

B.18 HINF: REQUESTING I/O HANDLER INFORMATION

```
          .DEFIN  HINF,LUN,EV
          CAL     .+2
          JMP     .+4
          3600
          EV+0
          .DEC; LUN; .OCT
          .ENDM
```

INDEX