HEWLETT
PACKARD

RTE-A Driver Designer's

Manual

Data Systems Division
1266 Kifer Road
Sunnyvale, CA 94086-5304

Manual Part No. 92077-90019
U0887

Printed in U.S.A. June, 1983
Update 3 August, 1987

Printing History

The Printing History below identifies the edition of this manual and any updates that are included. Periodically, update packages are distributed which contain replacement pages to be merged into the manual, including an updated copy of this printing history page. Also, the update may contain write-in instructions.

Each reprinting of this manual will incorporate all past updates; however, no new information will be added. Thus, the reprinted copy will be identical in content to prior printings of the same edition with its user-inserted update information. New editions of this manual will contain new information, as well as all updates.

To determine what manual edition and update is compatible with your current software revision code, refer to the Manual Numbering File or the Computer User's Documentation Index. (The Manual Numbering File is included with your software. It consists of an "M" followed by a five digit product number.)

First Edition	Feb 1982	
Second Edition	Jun 1983	
Update 1	Jan 1985	Clarif. of short DMA Transfers
Reprint	Jan 1985	Update 1 Incorporated
Update 2	Jan 1986	Manual Enhancement
Reprint	Jan 1986	Update 2 Incorporated
Update 3	Aug 1987	Error Code Chgs., #XQSB Update Rev. 5000 (Software Update 5.0)

Table of Contents

Chapter 1
Introduction

Chapter 2
System I/O Tables

Chapter 3
Device Driver

Chapter 4
Interface Driver

Chapter 5
General Driver Concerns

Chapter 6
Device and Interface Driver Interactions

Chapter 7
Callable System Routines

Chapter 8
Privileged Drivers

Chapter 9
I/O Card Programming

List of Illustrations

List of Tables

Chapter 1
Introduction

A program is allowed to do I/O (input/output) transfers only under the supervision of the operating system. While a user program is executing, the memory protect feature is on. This feature serves the dual function of protecting the operating system from inadvertant destruction by a user program and also insures that the operating system itself controls all I/O transfers. Any program that attempts an I/O instruction while the memory protect feature is on will cause an interrupt, suspending the program and transfering control to the system. The system then aborts the offending program.

All I/O requests are made to the system through EXEC calls, which are requests to the operating system. When a program which makes an EXEC request is loaded, the JSB EXEC instruction is replaced by an unimplemented instruction. The unimplemented instruction is trapped by the system and tested against an instruction chosen to represent the EXEC request. If it passes the test, and if the parameters in the request are valid, the request is processed. Otherwise, the program may be aborted. (It is possible to specify "no abort" in some cases.)

I/O requests are sorted out (through the request code) and processed by the operating system modules called RTIOA and IOMOD. These two modules work together, and are referred to jointly as RTIOA. One of the several functions of RTIOA is to relate the logical unit referenced by the user request to a physical device, which it does through the LUT (Logical Unit Table). The user request is then put into the form of a table called the I/O control block.

The information set up in the control block is processed for the I/O operation by operating system modules called drivers. Drivers may be divided into two modules; a module that deals with the device and a module that deals with the I/O card. Together they perform the single function of implementing the I/O request made by the user program and formatted by the system.

Prior to entering the driver, the system takes the information from the control block and puts it into another table, called the driver's DVT (Device Table). The request in the DVT is processed by the driver to perform the desired action (input, output or control). All information pertaining to the operation of a specific request is maintained in the DVT, which therefore becomes the primary directing force of driver operations.

Data is usually transferred under DMA (direct memory access). With DMA, an entire buffer is transmitted before the computer receives an interrupt signifying completion. Alternately, a driver may set up a card to create an interrupt per word/byte. In either case, once the transfer is set up, it proceeds on an interrupt basis. When the device is not ready to make an actual transfer, other processing takes place. When an interrupt occurs, the driver is entered (under system control) to take the proper action.

When a user request is received by the system, it may not be possible to initiate the operation immediately, since another request may be in progress. In this case the I/O control block is linked to previous blocks in a list. The list itself is linked to the DVT of the driver. When the driver is finished processing one request, the system sets it up for the next request.

I/O requests are generally linked to the driver in order of the priority of the program making the request.

User I/O Requests

The several requests associated with I/O are given below. Not all of these requests reach the driver. Those requests that are processed by the driver are indicated.

Function	Request	Code Seen by the Driver
READ	1	1
WRITE	2	2
CONTROL	3	3
STATUS	13	NONE
CLASS READ	17	1 (READ)
CLASS WRITE	18	2 (WRITE)
CLASS CONTROL	19	3 (CONTROL)
CLASS WRITE/READ	20	1 (READ)
CLASS GET	21	NONE
LU LOCK	NONE	NONE CALL LURQ (......)
CLEAR CLASS	NONE	NONE CALL CLRQ (......)

Figure 1-1. User I/O Requests

The driver processes only three basic requests:

1. Input
2. Output
3. Control

Consistent with this philosophy, class I/O requests also reduce to the basic three. The driver does have a means of identifying a class request through certain bits recorded in the DVT, but this is not normally a driver concern.

There are two types of status requests. The "static" status request (request code 13) does not cause the driver to be entered and so the actual device is not accessed. The status is that taken from the DVT and represents the status upon last I/O completion. The "dynamic" status is implemented through a control request (request code 3, subfunction 6) and thus causes the driver to be entered. The driver must recognize a subfunction code which differentiates the dynamic status request from other control requests.

Device-Interface Separation

A driver may be broken down into two parts (the device driver and the interface driver) or remain as a single driver (the interface driver). A device driver is not required, or useful, if the interface card is used to control a single device or identical devices. The device driver proves most useful when there are several possible device types cabled to the same interface type, e.g. the HP-IB.

The device driver, when present, formats the output buffers or interprets the incoming buffers according to the characteristics of the device. If the device driver is absent, as in minimum-sized systems, then the individual programs must perform the interpretation functions that would normally be done by the device driver.

The interface driver may perform some functions other than I/O, although they should not be device-specific. For example, it may append an EOR (end-of-record) character to output or it may translate all characters into a specific code (such as binary-coded decimal). It may recognize special characters and manipulate the data accordingly or it may transmit all characters unchanged.

One advantage of this approach is that the characteristics of the device may be changed by accessing the device driver only. For example, the number of lines/page on a line printer could be changed by a control request with a subfunction defining the number of lines. This could be done even if the interface card was busy with another device at the time.

Another advantage is the ease by which new devices may be connected to the I/O cards. Using an existing interface driver, one need only write the device driver to handle the characteristics of that new device.

Even though there are some advantages to separating the characteristics of the device and the interface and breaking the driver into two parts, this separation is not always recommended. For specialized I/O interface situations, it may be preferable that a single interface driver be designed to handle both the device and the interface.

I/O Request Interaction

Figure 1-2 (below) is a simplified representation of how the user request interacts with the drivers.

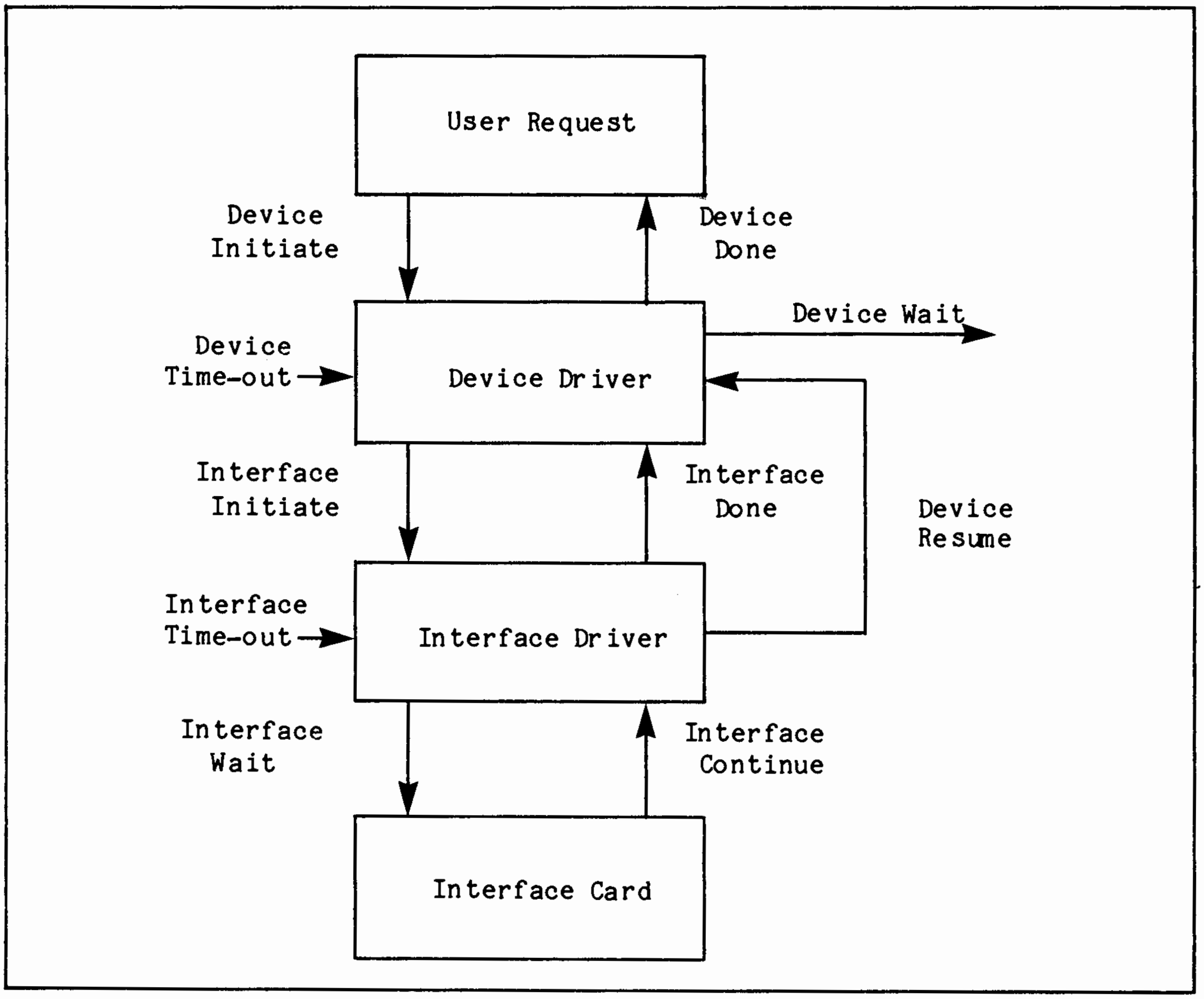

Figure 1-2. I/O Request Interaction

These interactions are described in more detail below. Although the diagram shows several reasons for entering a driver, the driver is always entered at the same place, whatever the reason. The reason for entering the driver is contained in a code in the A register. This code is the entry "directive." Exit from the the driver is to a different return point, according to the type of exit.

DEVICE INITIATE (DI): This is the starting point for processing all EXEC and XSIO calls, assuming that a device driver exists.

DEVICE DONE (DD): This exit completes the active request.

DEVICE RESUME (DR) calls the device driver to finish processing an asynchronous interrupt detected by the interface driver.

DEVICE WAIT (DW): This exit permits the device driver to await the completion of some timed action. It may also be used to indicate completion of asynchronous processing begun through a resume entrance.

DEVICE TIME-OUT (DTO): A request on the device has timed out (perhaps a failure). The device driver may use time-outs for the purpose of issuing periodic requests to the interface driver. For example, to check the status of a communications line.

INTERFACE INITIATE (II): This entry indicates the start of a request on the I/O card.

INTERFACE DONE (ID): This exit indicates the completion of a request started on the I/O card.

INTERFACE CONTINUE (IC): This is the continuation entry into the driver caused by an interrupt. This includes asynchronous interrupts. For example, an SRQ interrupt on the HPIB is an asynchronous interrupt.

INTERFACE WAIT (IW): The driver takes this exit to wait for an interrupt or timeout.

INTERFACE TIME OUT (ITO): This entry indicates an expected interrupt was not received in the alloted time.

"Asynchronous interrupt," as used in this manual, means an interrupt that occurs when the I/O card is not busy with a user or system I/O request. The I/O card is usually idle, and has been armed to recognize asynchronous interrupts, such as are generated when a user strikes a terminal key when no read is pending on the I/O card. An "expected interrupt," on the other hand, is one which signals the completion (or continuation) of a request from the system or a program.

When the system enters a driver, I/O interrupts are in a "hold-off" state. They are enabled only for the time base generator (TBG) and privileged drivers in the system. Since an entry to a non-privileged driver is always controlled by the system, a driver may call upon subroutines within the operating system. Several subroutines are supplied to make certain tasks easier for the driver and to avoid the duplication of functions from driver to driver.

There is no direct path between the device driver and the interface driver. For example, the device driver never directly calls the interface driver. Both modules are entered only by the system and each module returns to the system. Communication between the device driver and the interface driver may take place through a parameter area located in the DVT for that device.

A DVT always exists even though a device driver is not required. The request is always formatted in the DVT and status information passed back in the DVT, whether or not a device driver exists.

When a request is made by the user or the system, it is formatted into a table called the I/O control block and linked to the DVT. Several requests may already be in the linked list - the newest request is added on. When a request reaches the head of the initiation list, the system takes the information out of the control block and places it into the DVT. The normal flow is then as follows:

1. The device driver decodes the request which has been placed in its DVT and formats a request for the interface driver and stores it in the DVT. Upon exit, the device driver notifies the system that this is an "interface initiate" exit.

   The system takes the device drivers request and links it to the interface driver through the DVT.

2. When the request reaches the list head, the system enters the interface driver with a signal to begin the new request. The interface driver has a table called the IFT (InterFace Table) which links it to the driver's DVT. The interface driver picks up the request from the DVT and initiates an actual I/O sequence to the interface card. It then takes a "wait" exit to the system.

3. When an interrupt occurs, it is trapped by the system. The system enters the interface driver with a "continue" directive. Normally, the interrupt signifies the completion of a block transfer under DMA. In this case, the interface driver would post status information in the DVT and take a "done" exit.

4. The system then enters the device driver with a "continue" directive. The device driver may interpret the information received from the interface driver and reformat it for the device. It then indicates a "done" status and returns to the system.

5. If another control block is linked up to the DVT, its data is moved to the DVT and the device driver is entered for the next request.

Many complexities can arise in the above sequence. For example, a device driver may break a single user request into several requests upon the interface driver. Assume that the request is a disc read operation. This may be broken into at least two parts; a seek and then the actual read when the head reaches the proper cylinder. After the seek operation, the interface driver would be "done." But the device driver would not be done with the user request and hence would format another request to the interface driver.

The interface driver has no knowledge that it is doing a read operation or even that it is communicating with a disc drive. It is merely passing information back and forth. The only errors it handles are those dealing strictly with the interface card.

The device driver, on the other hand, knows that it is communicating to a disc and what control words or buffers are required for each request. It also knows to make certain checks on the parameters that are specific to the device. For example, it may check that a disc sector number is valid.

The "resume" exit from the interface driver and the "resume" entry into the device driver are used together, similiar to interface done and device continue. The resume is used when the interface driver has received an asynchronous interrupt which requires interpretation. Generally, this means that an interrupt has occurred from a device that was previously armed to recognise asynchronous interrupts. This may happen, for example, when a user strikes a key at a terminal to gain attention.

Driver Names and Module Type

The driver name is the symbol that is given in the NAM record in the source code. For example, ID.00 in the following:

```
MACRO,L
    NAM ID.00,0
```

The module type is 0 (zero) and is the parameter which follows the comma in the NAM record. Module type 0 identifies the driver as part of the operating system itself.

The convention for naming device and interface drivers is:

DDxnn is for Device Drivers
IDxyy is for Interface Drivers

x represents the Originator Code.
nn is the Device Driver Type, a number.
yy is the Interface Driver Type, a number.

When choosing the originator code, note that the period (.) and asterisk (*) and letters of the alphabet are reserved for drivers which orginate from Hewlett-Packard. Customers may use any of the following special symbols:

! " # $ % ^ _ ?

For example, DD$12 or ID!37. Other symbols are not legal in filenames.

The convention HP uses to refer to driver names has been changed from DD.nn to DD*nn. The HP driver relocatable filenames of the form %DD.nn have also been changed from %DD.nn to %DD*nn. HP driver names in the NAM statement (DD.nn) and driver entry points (DD.nn) remain the same.

DD*nn	Referenced driver name
%DD*nn	Driver relocatable name
DD.nn	Driver name in NAM statement
DD.nn	Driver entry point

Driver Type Codes

Device Driver

Device driver type codes are arranged by functional groupings as below. The type code is placed in DVT6 by the generator. The default by the generator is the field "nn" in the driver's entry point (as in DD.nn) but the type code can be changed at generation time. A user program may examine device type to determine what requests to issue. A multi-device driver can examine device type to determine what specific device to operate.

Category	Type	Device to be Driven
Keyboard Functions	00-05	Interactive point-to-point terminals
00-07 octal	07	Multipoint data link
System Peripherals	10B-11B	Plotters, graphics display
10-17 octal	12B-13B	Printers
	14B-17B	(reserved)
serial recording Devices	20B-24B	Mag tape, cassette
20-27 octal	25B	(reserved)
	26B	CS/80 tape
	27B	(reserved)
Random Recording Devices	30B	Floppy disc
30-36 octal	32B	ICD disc
	33B	CS/80 disc
	36B	PROM

HP-IB (37 octal)	37B	HP-IB interface bus
CPU Functions and Misc. Peripherals 40-47 octal	40B-43B 44B-47B	PROM I/O, WCS, powerfail, etc. Badge reader, strip printers, light pen, etc.
Digital/Analog 50-57 octal	50B-53B 54B-57B	Parallel interface card, etc. A/D, D/A
Data Communications 60-67 octal	60B-62B 63B-64B 65B-67B	Data comm., MUX, etc. (reserved) DS network, etc.
Instrument and Test 70-77 octal	70B-75B 76B-77B	Instruments Diagnostics

Interface Driver

The interface type for driver with entry point ID.yy defaults to "yy" and may be changed at generation time. Interface types are defined as follows:

00-07	Communication (hardwire or remote) interface cards, RS232
10-17B	Digital I/O cards
20-27B	Dedicated peripheral controller
30-37B	General purpose I/O card, e.g. HP-IB
40-47B	Special processor functions. ID.43 reserved for power fail
50-57B	Digital/Analog I/O
60-67B	Network Communications
70-77B	Instrument Controllers

Driver Entry Points

Entry points must agree with the name of the driver. Except for privileged drivers, the entry point should be the same as the driver name itself. For example:

```
MACRO,L
   NAM ID.00,0
   ENT ID.00
   ...
```

Privileged drivers have two entry points. For normal system entries, the entry point should be the same as for a standard driver. For privileged interrupt entry, use:

```
    PI.xx
```

For example:

```
    MACRO,R,L  * THIS IS THE START OF A PRIVILEGED DRIVER *
       NAM ID.51
       ENT ID.51,PI.51
       ...
```

GEN Pseudo Instruction

The assembler provides the capability of passing instructions from the source code to the generator. This is done with the GEN pseudo instructions, called "pseudo" because they do not produce actual CPU instructions.

For example:

```
    GEN 10,EID.37,QU:PR,TX:124
```

which specifies that the driver has entry point ID.37, priority queueing of requests and an extension area of 124 words. The number after the GEN instruction (in this case 10) indicates the number of words (2 characters/word) in the following string. The last character will be set to a space character if not specified.

With the exception of the "E" shown in the entry point name, all instructions in the GEN record are exactly the same format as would be given in the answer file to the generator. For example,

```
    GEN 10,DP:2:FM:GR:20040B:0
```

which sets driver parameters 2, 3, 4 and 5 to FM, GR, <two spaces>, and 0.

The GEN instructions provide default parameters for the driver and make it easier to prepare the answer file. Any parameters given in the answer file will override similiar commands given in the GEN instructions. For example, a different extension area size could be specified.

Any number of GEN instructions can appear in the driver.

Chapter 2
System I/O Tables

The system I/O tables provide an area of memory for storing and passing information about the I/O structure and I/O activity. These tables reside in and are maintained by the operating system. A summary of these tables is given below:

LUT	Logical Unit Table	Relates logical units to device tables.
DVT	Device Table	Maintains information about the I/O request and the physical device.
IFT	Interface Table	Maintains information for an interface card.
INTA	Interrupt Table	Relates interrupts from interface cards to interface tables.
MST	Map Set Table	Maintains information for correlating map sets to select codes.

Each of the tables above is built by the system generator. In some cases, as in the DVT and IFT, only part of the table is initialized by the generator. The contents of each table and how they are used will be discussed in this chapter.

There is a DVT entry for every device and an IFT entry for every interface card recognized by the system.

Normally, the only I/O tables referenced by the driver are the IFT and the DVT.

Figure 2-1 shows the interaction between the LUT, the DVTs and the IFTs.

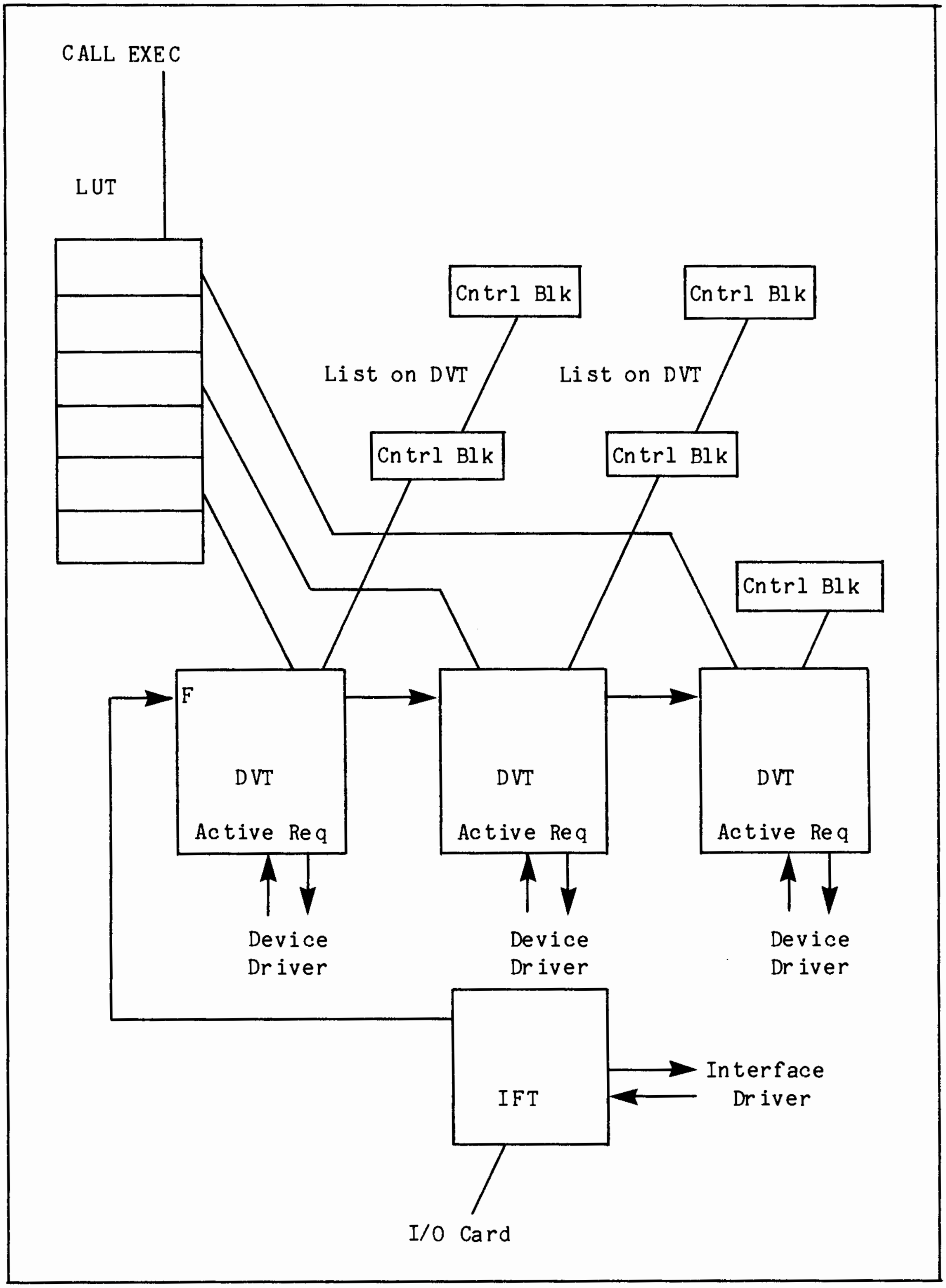

Figure 2-1. Request Lists on DVT and IFT

(There may be several IFTs in the system but only one is shown for clarity.)

Logical Unit Table LUT

The Logical Unit Table (LUT) is a variable length table built by the system generator. The LUT relates the logical unit (LU) in the user request (EXEC call) to the DVT. Its format is:

```
<address of some DVT>
<address of some DVT>
<address of some DVT>
     .
     .
     .
<address of some DVT>
```

Figure 2-2. Format of the Logical Unit Table

The logical unit is used as an index into the table. For example, for arbitrary logical unit LU X:

```
Pointer =  (X - 1) + address of LUT
DVT address  = contents of pointer
```

A pointer to the LUT and the number of entries in the LUT are globals located in RTIOA (see section on Table Pointers). The size of the LUT and its entries are set up by the system generator. The entries are modifiable on-line with an operator command and thus their direct use by any driver should be avoided, where possible.

More than one LU can point to the same DVT. An LU can also be assigned to zero (the bit bucket), in which case the corresponding entry in the LUT is zero.

Device Table DVT

The device table (DVT) is a variable-length table constructed by the system generator for each device in the system. It is the area from which the system communicates to the device driver information about the request. The system uses the DVT as a storage area for list link words, DVT status indicators and other system concerns. The device driver uses it for device dependent storage and a communication area to the interface driver. The interface driver may store device status in the DVT upon completion of a request.

Every device to be accessed by an LU must have a DVT. If no device driver exists, it is the responsibility of the interface driver to retrieve and post information in the DVT.

The format of the DVT is given in Figure 2-3.

<table>
<tr><th></th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>DVT1</td><td colspan="16">DVT Link Word</td></tr>
<tr><td>DVT2</td><td>Q</td><td colspan="15">Request Initiation List</td></tr>
<tr><td>DVT3</td><td>N</td><td colspan="15">Circular Node List</td></tr>
<tr><td>DVT4</td><td>P</td><td colspan="15">Circular DVT List</td></tr>
<tr><td>DVT5</td><td>X</td><td colspan="15">Address of Interface Table</td></tr>
<tr><td>DVT6</td><td colspan="2">AV</td><td colspan="6">Device Type</td><td colspan="7">Status</td><td>E</td></tr>
<tr><td>DVT7</td><td colspan="5">System Flags</td><td colspan="8">LU Lock Flag (Res #)</td><td>A</td><td colspan="2">RS</td></tr>
<tr><td>DVT8</td><td>B</td><td colspan="15">Buffer Limit Accumulator</td></tr>
<tr><td>DVT9</td><td>S</td><td colspan="7">(High-Low)/16</td><td colspan="8">Low Buff Limit/16</td></tr>
<tr><td>DVT10</td><td colspan="8">RESERVED</td><td colspan="8">Starting Physical Page</td></tr>
<tr><td>DVT11</td><td colspan="16">Timeout List Linkage</td></tr>
<tr><td>DVT12</td><td colspan="16">Device Driver Timeout Clock</td></tr>
<tr><td>DVT13</td><td colspan="16">Interface Driver Timeout Value</td></tr>
<tr><td>DVT14</td><td colspan="16">Device Driver Entry Address</td></tr>
<tr><td>DVT15</td><td colspan="2">TY</td><td>UE</td><td>Z</td><td colspan="6">Subfunction</td><td colspan="2">x</td><td>L</td><td>BB</td><td colspan="2">RQ</td></tr>
<tr><td>DVT16</td><td colspan="16">Request Parameter #1 / Error code with D,F</td></tr>
<tr><td>DVT17</td><td colspan="16">Request Parameter #2 / Transmission Log</td></tr>
<tr><td>DVT18</td><td colspan="16">Request Parameter #3 / Extended Status #1</td></tr>
</table>

Figure 2-3. Format of the Device Table

DVT19	Request Parameter #4 / Extended Status #2		
DVT20	I	Driver Communication	Device Priority
DVT21	# Driver Parameters	# Extension Words	
DVT22	DVT Extension Address		
DVT23	Starting Physical Page of Driver		
DVT24	M	Reserved	
DVT25	Spool Node List Pointer		
DVTP	Start of Driver Parameter Area		
	Start of DVT Extension Area (Storage)		

Figure 2-3 (continued). Format of the Device Table

DVT1 is the DVT link word, used by the system to put the DVT into various lists. For example, device driver requests passed to the interface driver (initiate exit) are linked via this word to word 3 of the IFT. It is set to -1 if not linked into any list.

DVT2 is the request initiation list. I/O control blocks built by the system as a result of a user request are linked to this word. It may be examined by a driver (device or interface) to determine if a request is currently in progress. Its contents will be 0 if no requests are pending.

> The Q bit is set to 0 by the generator as a default to indicate that the request list is ordered by priority. The driver may change Q to 1 to indicate a FIFO list is desired (first in, first out) via a GEN instruction.

DVT3 is the circular node list. This word links all DVTs which share a common node. A node connects all devices which cannot operate concurrently. An example is the keyboard/display and the mini-cassette drives on a 26XX terminal. Access to any of these features of the terminal excludes access to other features on that node at the same time. This list is set up by the generator.

> If a device on a node is busy, then the node itself is busy and no other devices on the node can be accessed. New requests are held off until the node is free. The N bit indicates the activity on the node. N=0 indicates node available; N=1 means the node is busy. The system sets this bit to 1 when it initiates a new request and it should not be changed by the driver.

When initiating a new request, the system checks the N bits on all the DVTs in the circular node list. One busy bit (there should not be more than 1) is sufficient to hold of the request until the node is not busy.

Devices in the node list share a common DVT extension, which is as large as the largest extension needed by any of the devices. The driver parameter area is not shared, but remains unique to each device.

If there is only a single DVT on the node, this word points to itself. Otherwise, it points to word 3 of the next DVT.

DVT4 is the circular DVT list. This word links all DVT's that point to the same IFT, i.e. it connects all device drivers with a common interface driver. It may be used by both the device drivers and the interface drivers. See the section on asynchronous I/O and polling in Chapter 6 for more information.

If there is only one DVT connected to the IFT (no circular list), then this word points to itself. Otherwise, it points to word 1 of the next DVT.

The P bit is for power fail. If the device driver wishes to handle power fail, it should set the P bit to 1. The generator defaults this bit to 0.

DVT5 is the IFT address. This is the address of the associated IFT. Bit 15 is reserved for future use and should not be changed by a driver.

DVT6 is Availability/Device Type/Status:

```
+-----------------------------------------------+
| AV  |  DEVICE TYPE     |       STATUS       |E |
+-----------------------------------------------+
```

AV is availablity. This is the current status of the DVT and is used by the system for I/O control. It may be examined by an operator command or by a driver to determine if a request is in progress.

AV	Meaning
00	The DVT is available for a new request to be initiated.
01	The associated device is "down." New requests will be I/O suspended.
10	The DVT is busy with a request. New requests may be pending, linked through DVT2.
11	The DVT is both down and busy.

DEVICE TYPE is a two-digit octal value used to describe the type of device associated with the DVT. The type is entered as a generator input or defaults to the driver number (see the section on Naming & Type Conventions).

A driver may use the type code to make decisions on what action to take in otherwise ambiguous situations.

In the absence of a device driver, the generator will default the device type to 70B.

STATUS is a general device status word reflecting the state of the device as posted by the driver upon last access. The bits have defined meaning as follows and should be so used by the driver:

```
+-----------------------------------------------+
|  7  |  6  |  5  |  4  |  3  |  2  |  1  |  0  |
|-----+-----+-----+-----+-----+-----+-----+-----|
| EOF | DB  | EOM | BOM | SE  | DF  | DF  |  E  |
|-----+-----+-----+-----+-----+-----+-----+-----|
|< - -  Set by driver, as needed  - - - ->| Set |
|                                         | by  |
|                                        System|
+-----------------------------------------------+
```

EOF is End Of File. Used for mini-cassette tapes, magnetic tapes, card readers, etc. EOF = 1 when condition is true.

DB is Device Busy. Indicates that the device is performing a function which prevents other operations from starting, e.g. mag tape rewind. DB = 1 when condition is true.

EOM is End Of Medium. Set when the current request has positioned (or will position) the physical medium past the maximum limit. For example, write 2 disc tracks when only 1 track remains to be used.

BOM is Beginning of Medium. When set, indicates that the medium is at the start of the recording area.

SE is Soft Error. An error occurred which caused the driver to attempt an error recovery operation. The E bit may or may not be set, depending upon whether or not the operation was eventually successful.

DF is Driver Definable.

E is an Error indicator set by system if the driver sets any error code in DVT16. Drivers should not change this bit.

DVT7 is System Flags/LU Lock Flag/Request Status:

```
+-----------------------------------------------+
| SYSTEM FLAGS | LU LOCK FLAG (ID #)     |A |  RS |
+-----------------------------------------------+
```

SYSTEM FLAGS are reserved for use by the operating system. These bits are updated by the system on each exit from the driver. The system flag bits are copied from bits 4 through 0 of the A register, which must be set by the driver prior to exit.

The meaning of these bits is given here in brief. They are covered more completely in the section on System-Driver Interface:

Type of	Bit Number				
Driver Exit	15	14	13	12	11
Done	0	0	0	H	T
Initiate	L	0	A	H	T
Wait	M	0	I	H	T

T = Set timeout on device request.

H = Hold off new device request.

A = Abort request on interface driver.

I = Report illegal resume entry.

L = Lock interface driver to this driver.

M = Maintain previous lock (if any).

LU LOCK FLAG is set by the system in response to an LU lock request. It consists of the ID segment number of the program which succeeded in gaining the lock. This field is zero if the device is not locked.

The A bit is a flag set by the system to indicate an abort is in progress. This flag will remain set from the time the device driver is notified of abort until abort processing is complete, at which time the bit will be set to 0 by the system.

RS, the Request State, is the status of the current DVT request. The driver may find it useful to examine the request state, for example, when it is called upon to abort the last request.

If 0, the DVT request is linked on the IFT. Interface driver proccesing on this request has not yet begun.

If 1, the DVT request is linked at the IFT head. It is currently being processed by the interface driver.

If 2, the DVT request is linked for interface done. The interface driver has completed the current DVT request. The driver will never see RS = 2.

If 3, the DVT request is linked for device done. The device driver has completed the current user request. The driver will never see RS = 3.

If there is no pending request (no list on DVT2), then the request state is invalid and should not be examined. This could occur, for example, if a "resume" entry is made into a terminal driver as a result of someone striking the keyboard.

DVT8 is Buffer Accumulator. If buffering is in effect, then this word is the total length of all buffered requests currently queued on the DVT. In addition, class requests, are always included in the accumulator.

The B bit (15) is set if the device is buffered.

DVT9 is Buffer Limits. This word stores the upper (HL) and lower (LL) buffer limits for the DVT. HL is a positive 16-bit value defining the limit above which requests will become suspended. LL is also a positive 16-bit value. When the accumulated count in DVT8 falls below LL, programs suspended for making a request when the accumulator was above the upper limit are allowed to repeat their requests.

To preserve table space, the values are stored as (HL-LL)/16 and LL/16. Buffer limits may also be changed by an operator command.

The S bit (15) is set if the device is buffer limited. When the limit is in effect, no new requests may be linked to the DVT. Programs which make buffered requests or class requests are, in this case, buffer-limit suspended.

DVT10 is the starting physical page of the partition containing the data for the I/O request. This page number is adjusted for system common when necessary.

DVT11 is Time-out List Linkage. This word is used to link all the DVTs and IFTs time-out clocks in a linked list. This list is ordered by time-out sequence, i.e. the DVT which could time out first appears first in the list.

The end of the list is terminated by zero (0) in word 11. If the DVT is not in the time-out list, then this word is set to minus one (-1).

DVT12 is the Time-out Clock. This word is a negative value, in tens of milliseconds, which is the running time-out clock for the device driver. This value plus any other time-outs before this one in the linked list is the current time-out value for a particular DVT.

DVT12 is initialized to 0 by the generator. The device driver must insert a negative value into DVT12 on each request if it wants time-out. In addition, it must set the "T" bit in the A register upon exit.

On the initiate exit, the time-out clock starts when the request is initiated on the interface driver. On the wait or done exit the clock starts when the exit is made. The clock is cleared on entry to the device driver.

DVT13 is a default time-out value for the interface driver when processing requests for this device.

The value is negative and given in tens of milliseconds. This value is put into IFT2 when the device driver request is initiated on the interface driver. The time-out clock starts when the interface driver returns to the system with the "T" bit in the A register set. It stops when the interface driver is reentered.

Time-out may be changed by the "TO" operator command.

DVT14 is Device Driver Address. This word is the address of the entry point for the associated device driver. This is 0 if no device driver exists. This address will not be used if the user request specifies that the device driver be bypassed (bit 15 set to 1 in user request).

DVT15 is Subfunction/Request Code. This word contains information about the user's request. Bits marked with an X are reserved for use by the system:

```
+-----------------------------------------------+
¦  TY ¦UE¦ Z¦    SUBFUNCTION    ¦ X  X¦L ¦BB¦ RQ ¦
+-----------------------------------------------+
```

SUBFUNCTION is derived from the ICNWD parameter of an EXEC request and provides control information about the request. The subfunction request is used differently according to whether the request is read/write (request code 1/2) or control (request code 3).

FOR READ OR WRITE REQUESTS (RQ = 1 or 2):

In order to provide device I/O transparency, particular control bits should be used to implement certain functions if applicable for the device. If these functions are not applicable for a device these bits, or combinations thereof may be used as desired.

Expansion of SUBFUNCTION for a read/write request:

11	10	9	8	7	6	Bit Number
DF	TR	DF	EC	DF	BI	Mnemonic

DF is driver definable.

TR, if 1/0, means transparency mode is/is not in effect. For nontransparency mode, terminators and/or imbedded control characters may be removed or added by the driver on input or output. An example is a the "CRLF" on a write to a CRT. When transparency mode is in effect, driver addition or removal of information is restricted. Refer to the DD.00 section of the Driver Reference Manual.

EC, if 1/0, indicates echo mode is/is not in effect. For echo mode the keyboard input is to be displayed as received. This is the normal mode of operation.

BI, if 1/0, means binary/ASCII information is to be transmitted. Refer to the DD.00 section of the Driver Reference Manual.

The subfunction bits should all be set to 1 if and only if the target device is a disc (type 30-37).

FOR CONTROL REQUESTS (RQ = 3)

For control requests, the SUBFUNCTION field should follow the conventions below. Note that "(Tape)" stands for a tape unit (cassette drives or mag tape).

Code	Action
00	Clear device
01	Write end-of-file (Tape)
02	Backspace one record (Tape)
03	Forward space one record (Tape)
04	Rewind (Tape)
05	Rewind standby (Tape)
06	Dynamic status
07	Set end-of media
10B	Set beginning of media
11B	List output line spacing (space no. of lines in positive optional parameters) or form feed (optional parameter is negative).
12B	Write gap (Tape)

13B	Forward space file (Tape)
14B	Backward space file (Tape)
15B	Conditional form feed
16B	Go to remote
17B	Go to local
20B	Enable program scheduling. Allows interrupt to schedule a program
21B	Disable (inhibit) scheduling of program
22B	Set time-out. The optional parameter is set as the new interface time-out interval
23B	Expect asynchronous interrupt (optional parameter = 0/1 = enable/disable)
24B	Set device address (subchannel)
25B	Driver definable
26B	Driver definable
27B	Driver definable
30-37B	Reserved for system expansion
40-77B	Driver definable

RQ is the request code:

RQ	Request type
0	Multibuffered
1	Read
2	Write
3	Control

TY, or request type, is additional information which is normally of no interest to the driver.

TY	Request type
0	Normal
1	Buffered
2	System
3	Class

The Z bit is the double buffer bit. If Z = 0, then DVT18 and DVT19 are simple parameters (no additional buffer). Z=1 designates that DVT18 is a second buffer address and DVT19 its length. This is applicable for read, write and control requests.

The UE bit is the user error bit. If the UE bit is set, the calling program is expected to process the device errors that occur. The program should examine status, and error returns in the A-Register and extended status which are accessible through a RMPAR call. The RMPAR call should be made to an unbuffered device. The UE is 0, the system provides normal error handling.

The UE bit is not functionally equivalent to the NS bit, which is described in the RTE-A Programmer's Reference Manual. Setting the UE bit only instructs the system to return error information to the calling program; the program is expected to process the returned error information.

The BB bit is set to bypass the device driver. If the user has specified in the request that the device driver not be called, then this bit is set to 1 by the system. This means that the interface driver is called, bypassing the device driver. The driver need not be aware of this bit.

> This bit will also be set if the device driver has been called for abort processing and has rejected the request as illegal.

The L bit is used by the system on read/write requests to indicate the source of data. 1 indicates that the data buffer is in the user or SAM map; 0 indicates the data is in the system map. This bit must be saved in its exact position in order for the driver to access any data in the buffer passed to it in DVT16 (see Chapter 7, Callable System Routines, for a description of $READ, $WRIT, $ONER, or $ONEW).

DVT16 is Request Parameter 1. This word serves two independent functions. On entering the driver, this word is the user's buffer starting address (RQ=1 or 2) or optional control parameter (RQ=3). (NOTE: The buffer address must be used in conjunction with the $READ, $ONER, $WRIT, or $ONEW subroutines to access data in the data buffer. It cannot be used as the absoulute address, however it can be used for calculating the relative address of any place in the buffer.) On exit, the driver reports error conditions in DVT16.

DVT17 is Request Parameter 2. This word serves two independent functions. Entering the driver, this word is the number of words (if positive) or characters (if negative) to be transmitted, or is an optional control parameter. On exit, the driver posts a positive transmission log in either words or characters depending on the original request. If a negative number of bytes was requested, a positive number of bytes is posted in the transmission log. The maximum range on these parameters is +32768 words (100000B) or -32767 bytes (100001B).

DVT18 is Request Parameter 3. This word may serve three distinct functions. If the Z bit in DVT15 is 0, then on an input request, it is another control parameter. If Z = 1 (control buffer), then DVT18 is a buffer address. This second buffer, in addition to DVT16, could be used for extended control information. The same rules for buffer access (see DVT16) apply.

> Upon returning to the system, DVT18 may contain device dependent error/status information. See the section (chapter 5) on Posting Errors for additional information.

DVT19 is Request Parameter 4. This word is like DVT18 except that if Z=1 it is the length in words (+) or characters (-) of the buffer at DVT18.

DVT20 contains the Initial Entry Flag, the Driver Communication Flags and the Device Priority:

> The I bit is set to 1 by the generator for use by the driver as a "first entry" flag. If the driver takes any special action on first entry, it should clear this bit so that the action is not repeated on subsequent entries.
>
> DRIVER COMMUNICATION FLAGS are nine bits through which the device and interface driver may pass information or maintain common status information which both drivers require. The generator will set these bits to zero.
>
> DEVICE PRIORITY (0-63) is the priority assigned to this DVT for linking purposes on the IFT. Default linking is FIFO (priority ignored) unless the interface driver changes its Q bit (IFT3, Figure 2-4) to specify priority linking.

DVT21 is the Number of Driver Parameters (bits 15 to 9) and Number of Extension Words (bits 8 to 0). The driver may wish to check the number of extension words assigned on first entry to ensure that it does not overlay an area of memory not available to it.

DVT22 is DVT Extension Address. This is the address of the first word of the DVT extension. The extension is a storage area for the device driver and should be used to store any temporary data needed to control a particular device. This extension lets a single device driver support several similiar devices.

> The DVT extension is not contiguous to the rest of the DVT.
>
> Devices linked together in the circular node list (DVT3) share a common extension. Therefore, drivers should not expect data in the extension area from a previous request to be valid.

DVT23 is the starting physical page number of the device driver if it was generated into a driver partition. The partitioned driver must be mapped into the system before being called. If the driver was not generated into a partition, DVT23 = 0.

DVTP is Driver Parameter Area. Driver parameters are configuration type variables for the device driver. They may be set at generation time or optionally by a driver control request. A typical driver parameter is the device HPIB address.

> The generator will set all driver parameters not specified at generation time to zero.

DVT24: If the M bit = 1, the current linked control block is located in SAM; otherwise, the M bit = 0. Bits 0-14 are reserved.

DVT25 points to the spool node list if the device is being spooled. If DVT25=0, the device is not being spooled.

Interface Table IFT

The interface table (IFT) is a variable length table constructed by the generator for each I/O card in the system. It is primarily a storage area for system I/O concerns, although the interface driver may examine the contents. The IFT extension is used by the interface driver for storage.

The format of the Interface Table (IFT) is shown below. In the discussion which follows, the generator-initialized values are indicated.

<table>
<tr><th></th><th>15</th><th>14</th><th>13</th><th>12</th><th>11</th><th>10</th><th>9</th><th>8</th><th>7</th><th>6</th><th>5</th><th>4</th><th>3</th><th>2</th><th>1</th><th>0</th></tr>
<tr><td>IFT1</td><td colspan="16">Time-out List Linkage</td></tr>
<tr><td>IFT2</td><td colspan="16">Time-out Clock</td></tr>
<tr><td>IFT3</td><td>Q</td><td colspan="15">Request List Linkage</td></tr>
<tr><td>IFT4</td><td colspan="16">Interface Driver Entry Address</td></tr>
<tr><td>IFT5</td><td colspan="16">Device Table Address ($DVT1)</td></tr>
<tr><td>IFT6</td><td colspan="2">AV</td><td colspan="6">Interface Type</td><td colspan="2">x x</td><td colspan="6">I/O select code</td></tr>
<tr><td>IFT7</td><td colspan="5">System Flags</td><td>F</td><td>M</td><td colspan="9"># Words IFT Extension</td></tr>
<tr><td>IFT8</td><td colspan="16">Starting Physical Page of Driver</td></tr>
<tr><td>IFT9</td><td>MA</td><td>X</td><td>ML</td><td>OH</td><td>MQ</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td>X</td><td colspan="5">Map Set Number</td></tr>
<tr><td>IFTX</td><td colspan="16">Start of IFT Extension (Storage)</td></tr>
<tr><td></td><td colspan="16">⋮</td></tr>
</table>

Figure 2-4. Format of the Interface Table

IFT1 is the Time-out List Linkage. DVTs and IFTs may be linked together in the time-out list. If this IFT is in the list, the contents of IFT1 point to the next IFT1 or DVT11. The list terminates in 0. If this IFT is not in the list, IFT1 is set to -1 (initial value by generator).

IFT2 is Time-out Clock. If active, this is a negative number indicating TBG ticks (10 millisecond intervals). It is not the actual time-out value when the time-out is active.

> Default time-out values for the interface driver are established in the DVT (see DVT13). The default value is stored in IFT2 upon entry to the interface driver. The interface driver can change the time-out value by changing IFT2.

IFT3 is Request List Linkage. If active, bits 14 to 0 are the address of word 1 on some DVT. If inactive, bits 14 to 0 are set to 0 (initial value by generator).

> The Q bit is defaulted to 1 by the generator to indicate a FIFO list (queue). If priority linking is desired, the default may be changed at generation time.

IFT4 is the Interface Driver Entry Address. Set by generator and not changeable.

IFT5 is Device Table Address. Set by generator to word 1 of some DVT. May be changed on-line by the LA (logical assignment) operator command. The result of the LA command may be to clear this word to zero if no DVTs remain assigned to the IFT.

> When the interface driver makes a resume or done exit, the system knows what device driver to enter to resume or continue the request by the contents of IFT5. Thus the interface driver may wish to control this word. See section on Asynchronous I/O and Polling.

IFT6 is Interface card characteristics.

> AV is Availability, the current status of the I/O interface. It is used by the system for I/O control.

AV	Meaning
00	IFT available
01	IFT locked to some DVT. No other DVTs can get to the head of the list until the lock is released.
10	IFT is busy.
11	IFT is busy and is locked.

INTERFACE TYPE identifies the type of I/O interface card which the interface driver is using. The generator defaults this value to the two octal digits within the interface driver name.

I/O SELECT CODE is set on the interface card itself by switches and is an input to the generator.

IFT7 contains the System Flags and the Extension Length.

SYSTEM FLAGS are five bits used to store temporary flags. It is used by the system and not needed by the driver. The bits are defined below in brief. They are covered more completely in the section on System-Driver Interface:

Type of	Bit Number				
Driver Exit	15	14	13	12	11
Done	Q	D	0	H	T
Continue	0	0	I	H	T
Resume	0	0	0	H	T

H = Hold off new interface driver request.
T = Set time-out on interface request.
I = Report illegal interrupt.
Q = Inhibit advance to next request on IFT3.
D = Defer entrance to the device driver.

The EXTENSION LENGTH may be checked by the driver to insure that it is sufficient. Otherwise the driver may overlay an area of memory not allocated to it. The length is an input to the generator.

The F bit is set to 1 by the generator for use by the driver as a "first entry" flag. If the driver takes any special action on first entry, it should clear this bit so that the action is not repeated on subsequent entries.

The M bit is set to 1 if the driver manages its own time-out queuing and dequeuing.

The bits marked "X" are reserved.

IFT8 is the starting physical page of the interface driver if the driver was generated into a driver partition. The partitioned driver must be mapped into the system before being called. If the ariver was not generated into a partition, IFT8 = 0.

IFT9 contains flags that deal with mapping I/O channels into map sets.

The MA bit, when set, indicates that a map set is allocated for this I/O channel.

The ML bit allows a map set to be locked to an I/O channel. When this bit is set, the system mapset deallocation routine, $MSRTN, will not deallocate the map set.

The OH bit is used by RTE-A to store the state of the hold flag when an I/O request is map-set suspended.

If the MQ bit is set, the I/O request associated with this IFT is on the map-set suspend queue.

The Map Set Number, if the MA bit is set, will be the number of the map set that is allocated for this I/O channel. If the MA bit is clear, this field will be meaningless.

The bits marked X are reserved.

IFTX is IFT Extension. The interface driver should use this area for storage of all temporary data associated with a particular I/O card. This area should also be used for any short DMA transfers instead of doing I/O directly from the driver code space. A single interface driver may use several IFTs, hence support several distinct (identical) I/O cards.

Interrupt Table INTA

The interrupt table is of variable length and built by the system generator. It consists of one word entries for each processor I/O select code (channel). The first entry is for select code 20. The one-word entries are defined as follows:

+IFT address	Address of the IFT corresponding to the interrupting select code. The system gains access to the interface driver via the IFT.
zero	Interrupt not expected (illegal) on this select code. Indicates generation or hardware failure.

Map Set Table MST

Map Set Table MST contains 24 words, each entry representing one of the map sets (numbered 8 through 31). The meaning of each entry depends on the state of bit 15:

Bit 15	Map Set	Meaning of Bits 14-0
0	Available	Pointer to next free map (0 if end of list).
1	Not Available	Pointer to IFT that is using map set.

Table Pointers

Global pointers are located in the system to permit access to the I/O tables. They are:

LOGICAL UNIT TABLE (LUT)

$LUTA Address of first word of logical unit table.

$LUT# Number of defined logical units (entries) within the table.

DEVICE TABLE (DVT)

$DVTA Address of first DVT table entry.

$DVT# Number of defined DVTs.

$DV1	Word	1
$DV2		2
$DV3		3
$DV4		4
$DV5		5
$DV6		6
$DV7		7
$DV8		8
$DV9		9
$DV10		10
$DV11		11
$DV12		12
$DV13		13
$DV14		14
$DV15		15
$DV16		16
$DV17		17
$DV18		18
$DV19		19
$DV20		20
$DV21		21
$DV22		22
$DV23		23
$DV24		24
$DV25	Word	25

Together, $DV1 through $DV25 specify the address of the word in the current DVT.

$DVTP Address of current DVT parameter area.

INTERFACE TABLE (IFT)

$IFTA Address of first IFT.

$IFT# Number of defined IFTs.

$IF1	Word	1	
$IF2		2	
$IF3		3	Together, $IF1 through $IF9
$IF4		4	specify the address of the
$IF5		5	current IFT.
$IF6		6	
$IF7		7	
$IF8		8	
$IF9	Word	9	
$IFTX	Address of current IFT extension.		

Each word in the current DVT and IFT can be accessed by adding a word count to a pointer to the first word, but the use of pointers to the words makes references to them easier to recognize in code, and it eliminates the need to use temporary variables to store the value of the word count plus the pointer.

The pointers to the current IFT or DVT are set up by the system prior to entering the driver.

INTERRUPT TABLE (INTA)

$INTA Contains a list of numbers used to find interface driver entry points for interrupt processing. When the system recognizes an interrupt from an interface card, it adds the interface card select code to the INTA entry for that select code to form the address of the interface driver interrupt entry point.

$INT# Number of defined entries in the interrupt table.

The method used by the system to index to the proper location in the interrupt table is:

```
LIB 4      GET INTERRUPTING SELECT CODE
.
.
.
ADB $INTA  INDEX TO PROPER ENTRY
```

The first user select code which can cause an interrupt is 20B.

If the value of the associated entry in the interrupt table is zero, then the message:

    Illegal interrupt from SCnn

is printed on the console.

MAP SET TABLE (MST)

$MST Contains the data for determining the current state of a map set.

$MST# Contains the total number of map sets (24).

$MSFRE Points to the first free entry on the map-set free list linked within the map set table ($MST). If no map sets are available, this entry will be zero.

$MSA Points to the location where map set number 0 ($MST-8) would be stored if it were in the map set table.

Chapter 3
Device Driver

System-Driver Interface

The system enters the device driver as indicated below. The address of the driver is picked up from the DVT.

All pointers to the DVT, as described in the chapter on System I/O Tables, are set prior to entering the driver. The registers and calling sequence are:

```
A-register = Entry Directive (bits 2-0)
B-register = DVT Address

JSB DD.XX
P+1  done
P+2  interface initiate
P+3  wait
```

On exit, bits 4-0 of the A-register are placed in bits 15-11 of DVT7 (the System Flags area).

The various entry directives and their codes in the A-register (binary) are:

Code	Meaning
000	Abort
001	Initiate
010	Continue
011	Time Out
100	Power Fail
101	Resume

The driver should mask off the high order bits of the A-register, as they are reserved for future changes.

The driver must increment its return address, stored at its entry point, to the proper exit as follows:

Source Code	Meaning	A-Register on Exit
JMP DD.nn,I P+1 return	Request complete on device driver.	0 0 0 H T
ISZ DD.nn JMP DD.nn,I P+2 return	Initiate request on interface driver.	L 0 A H T
ISZ DD.nn ISZ DD.nn JMP DD.nn,I P+3 return	Wait for resume entry from interface driver or device timeout.	M 0 I H T

Upon exit from the driver, bits 0-4 of the A-register are stored in the system flags area of DVT7. The meanings of the bits, if set, are:

T = Set timeout on device driver request.

H = Hold off new device driver request initiation.

A = Abort request on interface driver.

I = Report illegal resume entry.

L = Lock interface driver to this device driver.

M = Maintain previous lock (if any).

If the interface driver is locked, the DVT will remain at the head of the IFT upon an interface done exit. This prevents interleaving of requests from multiple DVTs on the same IFT.

The H bit holds off all initiate entries from RTE-A. Therefore, if the H bit is set on a done exit, the driver must plan on subsequent entry by device time-out or asynchronous resume.

Entry Directives

The system will set up the pointers to the DVT before entering the driver. If the device driver wishes to access the IFT, it should call the routine $DIOC (described in Chapter 7).

Upon entry, the directive code will be in the A-register bits 2-0. The driver should mask off the high order bits as they are reserved for future use. The DVT address will be in the B-register.

Initiate New Request

Upon entry, bits 2-0 of the A-register equal 001.

This entry is made when a new request is to be started. The request code (for read, write or control) is in DVT15 with parameters in DVT16 through DVT19. Additional information may be contained in the driver parameter area. The driver should first determine whether or not the request is applicable (for instance, a read request on a printer makes no sense) and, if not applicable, make an error exit.

Then the parameters may be checked against the device characteristics. For example, a read request to a disc may contain a track number that is outside the range. If so, the driver should make an error exit.

Illegal requests may be ignored by the driver by making an immediate normal completion exit.

If a legal request and device operation is required, the device driver formats one or more requests for the interface driver and makes an "interface initiate" exit. All information about the request to the interface driver must be contained in some area commonly agreed upon, typically the driver communication area in DVT20 or the DVT extension. Examination or modification of request data buffers should always take place through system supplied I/O system routines. (See the section on Mapping Considerations.)

Prior to making a request on the interface driver, the device driver may change the request by altering the request parameters. Or it may set bits in the driver communication area of DVT20 as flags to the interface driver.

Resume Interrupt Processing

Upon entry, bits 2-0 of the A-register equal 101.

This entry is made because the system has received a "resume" exit from the interface driver; the device driver is called to resume processing. A possible reason for the interrupt is that someone struck a key at a terminal upon which there was no pending read request.

Typically, the resume exit is used to distinguish an asynchronous interrupt from an expected interrupt, which uses the continue entry.

Continue Processing

Upon entry, bits 2-0 of the A-register = 010.

This entry is made to continue the processing of the current request which the device driver has made upon the interface driver (always synchronous with what the device driver is doing).

The interface driver is done with the request. The device may initiate another request upon the interface driver or complete the request by making a done exit.

The driver might make another request, for example, in the case of a device requiring some extended protocol. In the case of the HP terminals, before sending/receiving data to/from the terminal the driver first sends an ENQ (enquiry) character. When the terminal is able to respond, it sends back an ACK (acknowledge). The ENQ/ACK handshake is given to the interface driver as one request; when it completes, the actual output buffer is given to the interface driver in another request.

Timeout Processing

Upon entry, bits 2-0 of the A-register equal 011.

This entry is made when the clock in DVT12 completes the time-out period.

The time-out period for a device driver cannot be established at generation time. It is enabled by the driver setting the time-out bit in the A-register on exit and its value determined by the contents of DVT12 at that time. The value should be a negative number whose absolute value indicates the number of Time Base ticks desired. Each tick is .01 second and so the time in seconds is found by dividing the value by 100. DVT12 is cleared by the system prior to entering the driver.

The clock starts immediately upon done or wait exit. If the exit is to initiate a request on the interface driver, then the clock starts upon entry to the interface driver.

The action taken on time-out may vary greatly from device to device. For example, a communications terminal driver may wish to keep itself in the time-out list until a "line open" condition is detected. Thus, it might call the interface driver upon receiving time-out to detect the open condition. If not received, it would, again, set itself up for time-out.

Device time-out may easily be confused with the time-out of the interface request but it is not the same. The default time-out value for the interface is taken from DVT13, and is unique to each device request on the interface driver. An interface time-out causes entry into the interface driver, and a device time-out causes entry into the device driver.

Abort Request

Upon entry, bits 2-0 of the A-register equal 000.

The device driver may be called to abort the current request if an I/O request is in progress and the program is aborted. The device driver must terminate the request as rapidly as possible within the limits of the device.

Prior to entering the driver (device or interface) the "A" bit in DVT7 is set to indicate that abort processing is in progress. It will be reset when abort processing is completed by the drivers.

The device driver may find it useful to examine IFT5 (backward reference to current DVT), IFT6 (availability field) and DVT7 (device request status, RS) in order to decide what action is appropriate.

The request to be aborted may be in process by the interface driver (RS=1) or it may simply be in the list (RS=0). For example, a poll request may be active on several devices on the HP-IB.

There are several possible options open to the device driver. It may:

1. Initiate an abort request on the interface driver. The request will take precedence over any request on the interface driver now in progress for that device.

2. Defer abort processing until the request completes. An abort may not be in the best interest of the device being controlled.

3. Allow the system to be totally responsible for abort processing on the interface driver by rejecting the abort request as "illegal". Normally, a request which is rejected as illegal causes an error message but if the abort request is rejected by the device driver, no message is issued. The abort request is passed on to the interface driver.

If the device driver elects to take the "wait" exit, then the system will ensure that time-out is active. If no time-out is specified by the driver, then a default of 1 second will be supplied. The driver will be entered again at the end of the time-out, or at the completion of the request. If the time-out occurs, the driver may check the "A" bit in DVT7 to determine that abort processing is in progress.

Abort processing completes when the device driver makes a done exit.

Power-Fail Restart

Upon entry, bits 2-0 of the A-register are set to 100.

The device driver will be called on power-fail restart only if it has indicated that it should be called. The driver indicated that it should be called to process power-fail restarts by setting the P bit in DVT4. If the driver processes power-fail, then it will be called upon every power failure, but only if it was busy at the time the failure occurred.

Driver Exit

Upon driver exit, there are three concerns:

1. Setting of system flags through bits in the A-register.

2. Posting status in the DVT.

3. Posting any errors, in addition to status.

The system flags are set regardless of whether the exit is to indicate "done," "interface initiate" or "wait." However, status and errors are posted only on the done exit.

It is important to remember that the status of the transfer of data and any tranfer errors should be posted by the interface driver. The device driver handles only device-dependent status and errors.

The topics of status and error posting are common to both the device driver and the interface driver and so they are covered in the chapter on General Driver Concerns.

System Flags

The three possible exit sequences from the device driver are given below. For each exit, bits 4-0 of the A-register have the meaning indicated. The B-register is meaningless.

The system takes the contents of A-register bits 4 through 0 and places them in the system flags area of DVT7.

A-Register Bit:	4	3	2	1	0
P + 1 "Done"	0	0	0	H	T
P + 2 "Initiate"	L	0	A	H	T
P + 3 "Wait"	M	0	I	H	T

T means set time-out. If set, the system will enter the device driver in the time-out list. See Time-out Processing.

H means hold. If set, the system will delay calling the device driver to start a new request. The driver normally sets this bit only to allow it to process interrupts through the resume entry, with device time-out in effect.

A means abort. If set, the system will call the interface driver with an abort directive.

I indicates illegal resume entry. If set, the system will issue an error message of the form:

```
Illegal interrupt from LU nn
```

where nn is the current logical unit number pointing to the DVT.

The bit should only be set in the case of an illegal resume entry.

L means lock IFT to DVT. If set, the DVT will remain at the head of the IFT upon "interface done." This prevents interleaved requests from several device drivers on one interface driver. Not every driver will encounter situations where it is necessary to use this bit.

> A side benefit of the lock is that the DVT will not be unlinked and relinked to the interface driver as one request completes and another is initiated. Thus, if the device driver knows that it has several requests to execute at "high speed," it may lock the IFT to reduce overhead.

M means maintain lock. On subsequent exits from the driver, a previously locked IFT will remain locked only if this bit remains set to 1. Note that an IFT will never remain locked on a device done exit.

Sample Device Driver

This section contains a listing for a sample terminal driver. Many of the features of the driver are not explained in detail in the manual because they are not essential to the structure of the driver. That is, there are many different ways the same result could be achieved and this listing represents one programmer's approach.

Although this sample driver has been tested, it is not guaranteed to correspond to the code in any driver shipped with the system. It is included here only as an example.

```
            ASMB,Q,C
*
*    NAME:   DD.20
*    SOURCE: 92071-18084
*    RELOC:  92071-16084
*    PGMR:   T.A.L.
*
*   *************************************************************
*   * (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1980.  ALL RIGHTS    *
*   * RESERVED.  NO PART OF THIS PROGRAM MAY BE PHOTOCOPIED,     *
*   * REPRODUCED OR TRANSLATED TO ANOTHER PROGRAM LANGUAGE WITHOUT*
*   * THE PRIOR WRITTEN CONSENT OF HEWLETT-PACKARD COMPANY.       *
*   *************************************************************
*
*
                  NAM DD.20,0      92071-16084  REV.2326 <830519.1125>
*
*
                  ENT DD.20
                  EXT $DV6,$DV15,$DV16,$DV17,$DV18,$DV19,$DV22
                  EXT $DVTP,$CVT3,$CVT,$ONER,$ONEW,$DV1,.MVW
*
                  GEN 1,PA
                  GEN 19,EDD.20,TX:45,TO:3000,DT:20B,QU:FI    *2326TH*

                  GEN 2,DX:1
*
                  GEN 7,M2645:1,DP:1:1
                  GEN 7,M2645:2,DP:1:2
*
                  GEN 7,M264X:1,DP:1:1
                  GEN 7,M264X:2,DP:1:2
```

```
*
      000000  A     EQU 0
      000001  B     EQU 1
*
00000 000000  DD.20 NOP
00001 000030R       STA DIREC      SAVE DIRECTIVE
00002 001175R       JSB SETAD      SETUP EXTENSION ADDR PTR'S
00003 000030R       LDA DIREC      GET DIRECTIVE
00004 000025R       AND B7
00005 002002        SZA            ABORT?
00006 000013R       JMP GO         NO
*
* ABORT *
*
00007 001260R       STA DVX14,I    ZERO CHARACTER ACCUMULATOR
00010 000457R       LDA B4         CALL INTERFACE DRIVER
00011 001155R       JSB CEXIT      WITH ABORT CODE
00012 001130R       JMP DDCM2      DEVICE COMPLETE
*
*
00013 000022R GO    CPA B1         INITIATE?
00014 000031R       JMP INIT       YES
00015 000023R       CPA B2         CONTINUATION?
00016 001164R       JMP CONT       YES
00017 000024R       CPA B3         TIMEOUT?
00020 001130R       JMP DDCM2      YES, DEVICE COMPLETE
00021 001121R       JMP DDCOM      DEVICE COMPLETE
*
*
00022 000001  B1    OCT 1
00023 000002  B2    OCT 2
00024 000003  B3    OCT 3
00025 000007  B7    OCT 7
00026 177767  M9    DEC -9
00027 177765  M11   DEC -11
00030 000000  DIREC NOP            DIRECTIVE
*
*
* INITIATION *
*
00031 001240R INIT  LDA ESCC       GET <ESCc> LOCK KEYBOARD
00032 001246R       STA DVX4,I     SAVE IT
00033 001214R       LDA ESC&       GET <ESC&>
00034 001247R       STA DVX5,I     SAVE IT
00035 000010X       LDA $DVTP,I    GET CTU (1 OR 2)
00036 000230R       IOR PLU        MERGE <p60>
00037 001250R       STA DVX6,I     SAVE <p61 OR p62>
00040 000002X       LDA $DV15,I    GET SUBFUNCTION
00041 001261R       STA DVX15,I    SAVE IT
00042 000642R       JSB ASCWT      ASCII WRITE (SYSTEM ADDR. SPACE)
00043 001261R       LDA DVX15,I    GET RQ
```

```
00044 000024R        AND B3
00045 000024R        CPA B3         CONTROL REQUEST?
00046 000420R        JMP CNTRL      YES
00047 000001X        LDA $DV6,I     GET DEVICE STATUS
00050 000411R        AND LBYTE      REMOVE OLD STATUS
00051 000001X        STA $DV6,I
00052 001261R        LDA DVX15,I    GET SUBFUNCTION
00053 000227R        AND ECHO       REMOVE ECHO BIT 8
00054 001261R        STA DVX15,I    SAVE INITIAL SUBFUNCTION MINUS ECHO BIT
00055 000024R        AND B3         GET RQ
00056 000003X        LDB $DV16,I    GET BUFFER ADDR
00057 001244R        STB DVX2,I     SAVE INITIAL ADDR.
00060 001246R        LDB DVX4       GET ESC SEQUENCE ADDR.
00061 000003X        STB $DV16,I    SAVE IT
00062 000004X        LDB $DV17,I    GET XLOG
00063 001255R        STB DVX11,I    SAVE INITIAL XLOG (-CHARS OR +WORDS)
00064 006020         SSB            CHARACTERS?
00065 000070R        JMP *+3        YES, SAVE THEM
00066 007004         CMB,INB        NO, CONVERT TO
00067 005000         BLS            - CHARACTERS
00070 001245R        STB DVX3,I     SAVE -CHAR LENGTH
00071 000022R        CPA B1         READ REQUEST?
00072 000240R        JMP READ       YES
*
*
* WRITE REQUEST *
*
00073 001245R WRITE  LDA DVX3,I     GET -CHAR LENGTH
00074 003004         CMA,INA        MAKE CHARACTERS POSITIVE
00075 001261R        LDB DVX15,I    GET SUBFUNCTION
00076 005727         BLF,BLF
00077 005200         RBL
00100 006020         SSB            ASCII?
00101 002001         RSS            NO, CHARACTER LENGTH OK
00102 000023R        ADA B2         YES, ADD TWO TO LENGTH FOR 'CRLF'
00103 001120R        LDB M257
00104 000000         ADB A
00105 006020         SSB            LENGTH > 256?
00106 002003         SZA,RSS        ZERO XLOG?
00107 000454R        JMP ERROR      YES, ILLEGAL REQUEST ERROR
00110 002300         CCE            E=1 FOR DECIMAL
00111 000011X        JSB $CVT3      CONVERT +CHAR'S TO ASCII
00112 001232R        LDA DN
00113 000012X        IOR $CVT+1
00114 001251R        STA DVX7,I     SAVE <dSPACE OR NUMBER>
00115 000012X        LDA $CVT+2
00116 001252R        STA DVX8,I     SAVE <NUMBER> TO WRITE
00117 001233R        LDA W          GET <W>
00120 001253R        STA DVX9,I     SAVE <W>
00121 000027R        LDA M11
00122 000004X        STA $DV17,I    BUFFER LENGTH
```

```
00123 002404         CLA,INA        ALLOW TIMEOUT
00124 001155R        JSB CEXIT      INITIATE WRITE ESCAPE SEQUENCE
*
00125 001060R        JSB FPORT      FLUSH PORT BUFFERS FOR MUX
00126 001155R        JSB CEXIT      INITIATE CNTRL REQ. 26B FOR MUX
*
00127 001234R        LDB ENQ        GET ENQUIRY
00130 000006X RACK   STB $DV19,I    SAVE 'ENQ' OR ZERO
00131 001261R        LDA DVX15,I    GET RQ
00132 000024R        XOR B3         MAKE SURE ITS A ASCII READ
00133 000237R        AND SBIT       (SYSTEM ADDR. SPACE)
00134 000002X        STA $DV15,I    SAVE IT
00135 001262R        LDA DVX16      GET 1 BYTE READ ADDRESS
00136 000003X        STA $DV16,I    SAVE IT
00137 003400         CCA            BUFFER LENGTH
00140 000004X        STA $DV17,I    SAVE IT
00141 002400         CLA
00142 000005X        STA $DV18,I    ZERO ASIC CONTROL WORD
00143 002004         INA            ALLOW TIMEOUT
00144 001155R        JSB CEXIT      SEND 'ENQ', READ 'ACK'
*
00145 006400         CLB            CLEAR 'ENQ'
00146 001262R        LDA DVX16,I    GET BYTE READ
00147 000411R        AND LBYTE      REMOVE LOW BYTE
00150 000236R        CPA ACK        'ACK' RECEIVED?
00151 002001         RSS            YES, CONTINUE
00152 000130R        JMP RACK       NO, RETRY FOR ACK ONLY
*
00153 001261R        LDA DVX15,I    GET SUBFUNCTION
00154 000226R        AND CBIT7      ADD 'CRLF'
00155 000225R        IOR BIT8       SET 'DISABLE HANDSHAKE' BIT FOR MUX
00156 000002X        STA $DV15,I    (USER ADDR. SPACE)
00157 001244R        LDA DVX2,I     GET INITIAL BUFFER ADDRESS
00160 000003X        STA $DV16,I    SAVE IT
00161 001245R        LDA DVX3,I     GET INITIAL BUFFER LENGTH
00162 000004X        STA $DV17,I    SAVE IT
00163 002400         CLA
00164 000005X        STA $DV18,I    ZERO ASIC CONTROL WORD
00165 002004         INA            ALLOW TIMEOUT
00166 001155R        JSB CEXIT      INITIATE WRITE
*
00167 000004X        LDA $DV17,I    GET XLOG (+CHAR'S)
00170 001260R        STA DVX14,I    SAVE IN EXTENSION
00171 000204R        JSB STAT       SETUP FOR 2 CHAR READ
00172 001155R        JSB CEXIT      SEND DC1, READ 'S' OR 'F'
00173 006400         CLB            ZERO ERROR CODE
00174 001256R        LDA DVX12,I    GET COMPLETION STATUS
00175 000411R        AND LBYTE      REMOVE LOW BYTE
00176 000232R        CPA S          SUCCESSFUL?
00177 000202R        JMP DONE       YES (B=ERROR CODE)
00200 001260R        STB DVX14,I    NO, ZERO XLOG
```

```
00201 000662R        JSB DYST       GET DYNAMIC STATUS
*
00202 000003X DONE   STB $DV16,I    SETUP ERROR CODE
00203 001121R        JMP DDCOM      DEVICE COMPLETE
*
00204 000000  STAT   NOP            SETUP FOR 2 CHAR READ
00205 001256R        LDA DVX12      GET READ ADDRESS
00206 000003X        STA $DV16,I    SAVE IT
00207 000231R        LDA M2         BUFFER LENGTH
00210 000004X        STA $DV17,I    SAVE IT
00211 001261R        LDA DVX15,I    GET RQ
00212 000661R        AND RQASC      MAKE SURE ITS A ASCII READ
00213 002004         INA            RQ=1
00214 000237R        AND SBIT       (SYSTEM ADDR. SPACE)
00215 000002X        STA $DV15,I    SAVE IT
00216 002400         CLA
00217 000005X        STA $DV18,I    ZERO ASIC CONTROL WORD
00220 000407R        LDA DC1        SETUP DC1
00221 000006X        STA $DV19,I    IN OPTIONAL PARAMETER
00222 002404         CLA,INA        ALLOW TIMEOUT
00223 000204R        JMP STAT,I     RETURN
*
*
00224 000100  BIT6   OCT 100        BINARY BIT 6
00225 000400  BIT8   OCT 400        'DISABLE HANDSHAKE' BIT
00226 177577  CBIT7  OCT 177577     ADD 'CRLF'
00227 177377  ECHO   OCT 177377     ZERO ECHO BIT 8
00230 070060  PLU    OCT 70060      <p60>
00231 177776  M2     DEC -2
00232 051400  S      OCT 51400      <S>
00233 140001  ILREQ  OCT 140001     ILLEGAL REQUEST
00234 017015  RS.CR  OCT 17015      RECORD SEPERATOR CARRIDGE RETURN
00235 006415  CR.CR  OCT 6415       CARRIAGE RETURN CARRIAGE RETURN
00236 003000  ACK    OCT 3000       'ACKNOWLEDGE'
00237 177767  SBIT   OCT 177767     ZERO S BIT (BIT 3)
*
*
* READ REQUEST *
*
00240 000024R READ   LDA B3
00241 006003         SZB,RSS        ZERO XLOG?
00242 000525R        JMP FSRF       YES, FORWARD SPACE ONE RECORD
00243 001235R        LDA S2         GET <s2>
00244 001251R        STA DVX7,I     SAVE IT
00245 001236R        LDA R          GET <R>
00246 001252R        STA DVX8,I     SAVE IT
00247 000026R        LDA M9         BUFFER LENGTH
00250 000004X        STA $DV17,I    SAVE IT
00251 002404         CLA,INA        ALLOW TIMEOUT
00252 001155R        JSB CEXIT      SEND READ ESCAPE SEUQENCE
*
```

```
00253 001060R       JSB FPORT      FLUSH PORT BUFFERS FOR MUX
00254 001155R       JSB CEXIT      INITIATE CNTRL REQ. 26B FOR MUX
*
00255 000407R       LDA DC1        SETUP FOR
00256 000006X READ5 STA $DV19,I    DC1 CODE IN UPPER BYTE
00257 001261R       LDA DVX15,I    GET INITIAL SUBFUNCTION
00260 000224R       IOR BIT6       SET BINARY BIT
00261 000237R       AND SBIT       (SYSTEM ADDR. SPACE)
00262 000002X       STA $DV15,I    SAVE IT
00263 001262R       LDA DVX16      GET DRIVER EXTENSION ADDR
00264 000003X       STA $DV16,I    SAVE IT
00265 000654R       LDA M5         BUFFER LENGTH
00266 000004X       STA $DV17,I    SAVE IT
00267 000410R       LDA B1415      SETUP FOR SPECIAL CHAR (CR)
00270 000005X       STA $DV18,I    IN ASIC CONTROL WORD
00271 002404        CLA,INA        ALLOW TIMEOUT
00272 001155R       JSB CEXIT      SEND DC1, READ 5 BYTES
*
00273 002400        CLA            ZERO ASIC CONTROL WORD
00274 000004X       LDB $DV17,I    GET XLOG (+CHARS)
00275 000022R       CPB B1         ASYNCHRONOUS INTERRUPT RECEIVED?
00276 000256R       JMP READ5      YES, TRY AGAIN (REQ. FOR MUX)
00277 001261R       LDB DVX15,I    GET INITIAL SUBFUNCTION
00300 000002X       STB $DV15,I    (USER ADDR. SPACE)
00301 001262R       LDB DVX16,I    GET LAST CHARACTERS READ
00302 000234R       CPB RS.CR      RSCR?
00303 000414R       JMP ZEROL      YES, END OF READ
00304 000235R       CPB CR.CR      CRCR?
00305 000414R       JMP ZEROL      YES, RETURN KEY STRUCK
00306 000005X       STA $DV18,I    SAVE ASIC CONTROL WORD
00307 001244R       LDA DVX2,I     GET INITIAL BUFFER ADDR
00310 000003X       STA $DV16,I    SAVE IT
00311 001262R       LDA DVX16,I    GET FIRST AND SECOND BYTES
00312 001263R       LDB DVX17,I    GET THIRD AND FOURTH BYTES
00313 005700        BLF            MERGE THE FOUR
00314 100104        RRL 4          BYTES IN ORDER
00315 005700        BLF            TO FIND
00316 100104        RRL 4          BUFFER LENGTH
00317 001700        ALF
00320 101104        RRR 4
00321 003007        CMA,INA,SZA,RSS BUFFER LENGTH ZERO?
00322 000414R       JMP ZEROL      YES, READ STATUS
00323 000001        STA B          SAVE LENGTH
00324 007004        CMB,INB        MAKE LENGTH POSITIVE (+CHAR'S)
00325 103101        CLO            CLEAR OVERFLOW
00326 001257R       STB DVX13,I    SAVE REQUEST LENGTH (+CHARS)
00327 001245R       ADB DVX3,I     ADD ORIGIONAL LENGTH (-CHAR'S)
00330 102301        SOS            SKIP OVERFLOW SET
00331 006021        SSB,RSS        REQUEST LENGTH >= BUFFER LENGTH?
00332 001245R       LDA DVX3,I     YES, USE BUFFER LENGTH
00333 000004X       STA $DV17,I    SAVE LENGTH (-CHARS)
```

```
00334 006020        SSB             REMAINING LENGTH POSITIVE?
00335 006400        CLB             NO, ZERO INTERRUPTS TO BIT BUCKET
00336 000407R       LDA DC1         DC1 IN UPPER BYTE
00337 000001        IOR B           MERGE REMAINING INTERRUPTS TO BIT BUCKET
00340 000006X       STA $DV19,I     SAVE DC1 + INTERRUPTS TO BIT BUCKET
00341 002404        CLA,INA         ALLOW TIMEOUT
00342 001155R       JSB CEXIT       SEND DC1, READ DVT17 BYTES
*
00343 000002X       LDA $DV15,I     GET SUBFUNCTION
00344 101046        LSR 6
00345 000004X       LDB $DV17,I     GET XLOG (+CHARS)
00346 000010        SLA             ASCII?
00347 000357R       JMP XLOG        NO, DO NOT ADJUST XLOG
00350 001257R       LDA DVX13,I     YES, GET REQUEST LENGTH (+CHARS)
00351 000405R       ADA M1          SUBTRACT ONE
00352 001245R       ADA DVX3,I      ADD BUFFER LENGTH (-CHARS)
00353 002003        SZA,RSS         (RL-1) = BL?
00354 000405R       ADB M1          YES, XLOG = XLOG -1
00355 002020        SSA             (RL-1) < BL?
00356 000231R       ADB M2          YES, XLOG = XLOG - 2
00357 006020  XLOG  SSB             XLOG NEGATIVE?
00360 006400        CLB             YES, ZERO XLOG
00361 001260R       STB DVX14,I     SAVE XLOG (+CHAR'S)
00362 004065        CLE,ERB         E=0/1, ODD/EVEN
00363 001266R       ADB DVX20,I     FIND LAST CHAR ADDR.
00364 002041        SEZ,RSS         LAST CHAR EVEN?
00365 001121R       JMP DDCOM       NO, DEVICE COMPLETE
00366 001213R       STB TEMP        SAVE CHARACTER ADDR PTR
00367 000013X       JSB $ONER       YES, GET LAST WORD
00370 101261R        DEF DVX15,I
00371 100015X        DEF $DV1,I
00372 000411R       AND LBYTE       REMOVE LOWER BYTE (SPEC CHAR)
00373 000002X       LDB $DV15,I     GET SUBFUNCTION
00374 005727        BLF,BLF
00375 005200        RBL
00376 006021        SSB,RSS         BINARY?
00377 000406R       IOR B40         NO, PAD WITH A BLANK
00400 001213R       LDB TEMP        GET CHARACTER ADDR PTR
00401 000014X       JSB $ONEW       RESTORE WORD
00402 101261R        DEF DVX15,I
00403 100015X        DEF $DV1,I
00404 001121R       JMP DDCOM       DEVICE COMPLETE
*
00405 177777  M1    DEC -1
00406 000040  B40   OCT 40
00407 010400  DC1   OCT 10400       DC1 CODE IN UPPER BYTE
00410 140000  B1415 OCT 140000      SPECIAL CHAR (CR)
00411 177400  LBYTE OCT 177400      LOWER BYTE MASK
*
*
* ZERO LENGTH READ/DYNAMIC STATUS SETUP
```

```
*
00412 000232R TICST LDA S          GET <S>
00413 002001        RSS
00414 001260R ZEROL STA DVX14,I    ZERO XLOG
00415 001256R       STA DVX12,I    SAVE <S> OR NON <S>
00416 000662R       JSB DYST       GET DYNAMIC STATUS
00417 000202R       JMP DONE       DONE (B=ERROR CODE)
*
*
* CONTROL REQUEST *
*
00420 001261R CNTRL LDA DVX15,I    GET
00421 101046        LSR 6          SUBFUNCTION
00422 000467R       AND B77
00423 002003        SZA,RSS        RESET CTU?
00424 000503R       JMP RW         YES, DO REWIND
00425 000022R       CPA B1         WRITE EOF?
00426 000503R       JMP RW         YES
00427 000023R       CPA B2         BACKSPACE 1 RECORD?
00430 000530R       JMP BSRF       YES
00431 000024R       CPA B3         FORWARD SPACE 1 RECORD?
00432 000525R       JMP FSRF       YES
00433 000457R       CPA B4         REWIND?
00434 000503R       JMP RW         YES
00435 000460R       CPA B5         REWIND?
00436 000503R       JMP RW         YES
00437 000461R       CPA B6         DYNAMIC STATUS?
00440 000412R       JMP TICST      YES
00441 000462R       CPA B10        WRITE EOF IF NOT PREV. WRITTEN
00442 000473R       JMP EOF        YES
00443 000463R       CPA B13        FORWARD SPACE 1 FILE?
00444 000525R       JMP FSRF       YES
00445 000464R       CPA B14        BACKSPACE 1 FILE?
00446 000530R       JMP BSRF       YES
00447 000465R       CPA B26        WRITE END OF DATA (EOD)?
00450 000503R       JMP RW         YES
00451 000466R       CPA B27        LOCATE ABSOLUTE FILE IPRAM1?
00452 001072R       JMP ABSF       YES
00453 002401  ZERR  CLA,RSS        ZERO ERROR CODE
00454 000233R ERROR LDA ILREQ      ILL. REQ. DON'T DOWN/DO FLUSH
00455 000003X       STA $DV16,I    SAVE ERROR CODE
00456 001130R       JMP DDCM2      DEVICE COMPLETION
*
00457 000004  B4    OCT 4
00460 000005  B5    OCT 5
00461 000006  B6    OCT 6
00462 000010  B10   OCT 10
00463 000013  B13   OCT 13
00464 000014  B14   OCT 14
00465 000026  B26   OCT 26
00466 000027  B27   OCT 27
```

```
00467 000077  B77   OCT 77
00470 000200  B200  OCT 200
00471 000320  B320  OCT 320
00472 007700  B7700 OCT 7700
*
* END OF FILE (FUNCTION CODE = 10) *
*
00473 000232R EOF   LDA S          GET <S> IN UPPER BYTE
00474 001256R       STA DVX12,I    SET TO SUCCESSFUL
00475 000662R       JSB DYST       GET DYNAMIC STATUS
00476 000001X       LDA $DV6,I     GET DEVICE STATUS
00477 000471R       AND B320
00500 002002        SZA            AT EOF, LP, OR REWINDING?
00501 000453R       JMP ZERR       YES, DO NOT WRITE EOF
00502 002404        CLA,INA        WRITE EOF
*
* REWIND/WRITE EOF/WRITE EOD (FUNCTION CODE = 1,4,5 OR 26) *
*
00503 001216R RW    LDB U0         REWIND
00504 000022R       CPA B1         WRITE EOF?
00505 001221R       LDB U5         YES
00506 000465R       CPA B26        WRITE EOD?
00507 001222R       LDB U6         YES
00510 001251R       STB DVX7,I     SAVE <u0 OR u5 OR u6>
00511 001217R       LDB C
00512 001252R       STB DVX8,I     SAVE "C"
00513 000026R       LDB M9
00514 000004X       STB $DV17,I    BUFFER LENGTH
00515 001246R SEND  LDB DVX4       GET ESCAPE SEQUENCE ADDR.
00516 000003X       STB $DV16,I    SAVE IT
00517 002404        CLA,INA        ALLOW TIMEOUT
00520 001155R SEND1 JSB CEXIT      INITIATE REQUEST
*
00521 000204R       JSB STAT       SETUP FOR 2 CHAR READ
00522 001155R       JSB CEXIT      SEND DC1, READ 'S' OR 'F'
00523 000662R       JSB DYST       GET DYNAMIC STATUS
00524 000202R       JMP DONE       DONE (B=ERROR CODE)
*
* FORWARD/BACKWARD SPACE N RECORD/FILE (FUNCTION CODE = 2,3,13 OR 14) *
*
00525 001224R FSRF  LDB ONEP       FORWARD SPACE ONE RECORD/FILE
00526 000616R       JSB FBRF       SETUP ESCAPE SEQUENCE
00527 000520R       JMP SEND1      DO IT
*
00530 001224R BSRF  LDB ONEP       BACKSPACE ONE RECORD/FILE
00531 000616R       JSB FBRF       SETUP ESCAPE SEQUENCE
00532 001155R       JSB CEXIT      DO IT
*
00533 000204R       JSB STAT       SETUP FOR 2 CHAR READ
00534 001155R       JSB CEXIT      SEND DC1, READ 'S' OR 'F'
00535 000662R       JSB DYST       GET DYNAMIC STATUS
```

```
*
00536 000001X       LDA $DV6,I     GET STATUS
00537 001043R       AND BIT4       GET LOAD POINT BIT
00540 002002        SZA            AT LOAD POINT?
00541 000202R       JMP DONE       YES, DONE (B=ERROR CODE)
00542 000001X       LDA $DV6,I     GET STATUS
00543 001044R       AND BIT7       GET EOF BIT
00544 002003        SZA,RSS        AT EOF?
00545 000571R       JMP RECFL      NO, CHECK FOR RECORD OR FILE
00546 000642R BS2R  JSB ASCWT      ASCII WRITE (SYSTEM ADDR. SPACE)
00547 001225R       LDB TWOP
00550 000023R       LDA B2         BACKSPACE TWO RECORDS
00551 000616R       JSB FBRF       SETUP ESCAPE SEQUENCE
00552 001155R       JSB CEXIT      DO IT
*
00553 000204R       JSB STAT       SETUP FOR 2 CHAR READ
00554 001155R       JSB CEXIT      SEND DC1, READ 'S' OR 'F'
00555 000662R       JSB DYST       GET DYNAMIC STATUS
*
00556 000001X       LDA $DV6,I     GET STATUS
00557 001043R       AND BIT4       GET LOAD POINT BIT
00560 002002        SZA            AT LOAD POINT?
00561 000202R       JMP DONE       YES, DONE (B=ERROR CODE)
00562 000001X       LDA $DV6,I     GET STATUS
00563 001044R       AND BIT7       GET EOF BIT
00564 002002        SZA            AT EOF?
00565 000202R       JMP DONE       YES, DONE (B=ERROR CODE)
00566 000642R       JSB ASCWT      ASCII WRITE (SYSTEM ADDR. SPACE)
00567 000024R       LDA B3         FORWARD SPACE ONE RECORD
00570 000525R       JMP FSRF       DO IT
*
00571 001261R RECFL LDA DVX15,I    GET INITIAL SUBFUNCTION
00572 000472R       AND B7700
00573 000470R       CPA B200       BACKSPACE ONE RECORD?
00574 000202R       JMP DONE       YES, DONE (B=ERROR CODE)
00575 000642R       JSB ASCWT      ASCII WRITE (SYSTEM ADDR. SPACE)
00576 001224R       LDB ONEP       NO, THEN FORWARD SPACE
00577 000024R       LDA B3             ONE RECORD
00600 000616R       JSB FBRF       SETUP ESCAPE SEQUENCE
00601 001155R       JSB CEXIT      DO IT
*
00602 000204R       JSB STAT       SETUP FOR 2 CHAR READ
00603 001155R       JSB CEXIT      SEND DC1, READ 'S' OR 'F'
00604 000662R       JSB DYST       GET DYNAMIC STATUS
*
00605 000642R       JSB ASCWT      ASCII WRITE (SYSTEM ADDR. SPACE)
00606 001224R       LDB ONEP       BACKSPACE ONE FILE
00607 000464R       LDA B14
00610 000616R       JSB FBRF       SETUP ESCAPE SEQUENCE
00611 001155R       JSB CEXIT      DO IT
*
```

```
00612 000204R        JSB STAT       SETUP FOR 2 CHAR READ
00613 001155R        JSB CEXIT      SEND DC1, READ 'S' OR 'F'
00614 000662R        JSB DYST       GET DYNAMIC STATUS
*
00615 000546R        JMP BS2R       BACKSPACE TWO RECORDS
*
00616 000000  FBRF   NOP            SPACE N RECORDS/FILES
00617 001252R        STB DVX8,I     SAVE <Np>
00620 001223R        LDB UFRWD
00621 000023R        CPA B2         FORWARD SPACE?
00622 001227R        LDB UBKWD      NO, BACKSPACE
00623 000464R        CPA B14
00624 001227R        LDB UBKWD      BACKSPACE
00625 001251R        STB DVX7,I     SAVE <u+ OR u->
00626 001226R        LDB ONEC
00627 000463R        CPA B13        RECORD?
00630 001230R        LDB TWOC       NO, FILE
00631 000464R        CPA B14
00632 001230R        LDB TWOC       FILE
00633 001253R        STB DVX9,I     SAVE <1C OR 2C>
00634 000657R        LDB M12
00635 000004X        STB $DV17,I    BUFFER LENGTH
00636 001246R        LDB DVX4       GET ESCAPE SEQUENCE ADDR.
00637 000003X        STB $DV16,I    SAVE IT
00640 002404         CLA,INA        ALLOW TIMEOUT
00641 000616R        JMP FBRF,I     RETURN
*
* ASCII WRITE SUBROUTINE *
*
00642 000000  ASCWT NOP
00643 001261R        LDA DVX15,I    GET SUBFUNCTION
00644 000661R        AND RQASC      CLEAR BITS 6,7,8 & RQ
00645 000660R        IOR B602       MAKE SURE ITS A ASCII WRITE
00646 000237R        AND SBIT       (SYSTEM ADDR. SPACE)
00647 000002X        STA $DV15,I    INHIBIT 'CRLF'
00650 002400         CLA
00651 000005X        STA $DV18,I    ZERO ASIC CONTROL WORD
00652 000006X        STA $DV19,I    ZERO OPTIONAL PARAMETER
00653 000642R        JMP ASCWT,I    RETURN
*
00654 177773  M5     DEC -5
00655 177771  M7     DEC -7
00656 177770  M8     DEC -8
00657 177764  M12    DEC -12
00660 000602  B602   OCT 602        ASCII WRITE, INHIBIT ENQ-ACK FOR MUX
00661 177074  RQASC  OCT 177074     ZERO BITS 6,7,8 & RQ
*
* DYNAMIC STATUS (FUNCTION CODE = 6) *
*
00662 000000  DYST   NOP
00663 000662R        LDA DYST       STORE RETURN ADDRESS
```

```
00664 001267R        STA DVX21,I      AT DVX21
00665 000642R        JSB ASCWT        ASCII WRITE (SYSTEM ADDR. SPACE)
00666 001237R        LDA UP
00667 001251R        STA DVX7,I       SAVE <^>
00670 000655R        LDA M7           BUFFER LENGTH
00671 000004X        STA $DV17,I      SAVE IT
00672 001246R        LDA DVX4         GET ESCAPE SEQUENCE ADDR.
00673 000003X        STA $DV16,I      SAVE IT
00674 002404         CLA,INA          ALLOW TIMEOUT
00675 001155R        JSB CEXIT        SEND STATUS ESCAPE SEQUENCE
*
00676 000002X        LDA $DV15,I      MAKE SURE
00677 000024R        XOR B3           ITS A
00700 000002X        STA $DV15,I      ASCII READ
00701 001262R        LDA DVX16        GET READ ADDR.
00702 000003X        STA $DV16,I      SAVE IT
00703 000656R        LDA M8           BUFFER LENGTH
00704 000004X        STA $DV17,I      SAVE IT
00705 000407R        LDA DC1          SETUP DC1 CODE
00706 000006X        STA $DV19,I      IN OPTIONAL PARAMETER
00707 002400         CLA
00710 000005X        STA $DV18,I      ZERO ASIC CONTROL WORD
00711 002004         INA              ALLOW TIMEOUT
00712 001155R        JSB CEXIT        SEND DC1, READ 8 BYTES STATUS
*
00713 000001X        LDA $DV6,I       GET DEVICE STATUS
00714 000411R        AND LBYTE        REMOVE OLD STATUS
00715 000001X        STA $DV6,I
*
00716 001264R        LDA DVX18,I      GET STATUS BYTES 0 & 1
00717 001265R        LDB DVX19,I      GET STATUS BYTE 2
00720 005700         BLF              MERGE THE
00721 100110         RRL 8            THREE BYTES
00722 001700         ALF              TO FORM
00723 101110         RRR 8            STATUS WORD
00724 001054R        AND B7777        REMOVE UPPER FOUR BITS
00725 000005X        STA $DV18,I      SAVE STATUS WORD
*
* EXAMINE STATUS *
*
00726 000457R        LDB B4           SET BIT 2
00727 000224R        AND BIT6         GET WRITE PROTECT BIT
00730 002002         SZA              WRITE PROTECT?
00731 001036R        JSB DV6ER        YES, SET 'WP' IN DV6
00732 000462R        LDB B10          SET BIT 3
00733 000005X        LDA $DV18,I      GET STATUS WORD
00734 000462R        AND B10          GET SOFT ERROR BIT
00735 002002         SZA              SOFT ERROR?
00736 001036R        JSB DV6ER        YES, SET 'SE' IN DV6
00737 000224R        LDB BIT6         SET BIT 6
00740 000005X        LDA $DV18,I      GET STATUS WORD
```

```
00741 001043R        AND BIT4        GET TAPE BUSY BIT
00742 002002         SZA             TAPE BUSY?
00743 001036R        JSB DV6ER       YES, SET 'DB' IN DV6
00744 000005X        LDA $DV18,I     GET STATUS WORD
00745 000010         SLA             TAPE INSERTED?
00746 000753R        JMP CON         YES, CONTINUE
00747 000001X        LDA $DV6,I      GET DEVICE STATUS
00750 000023R        IOR B2          SET 'OF' IN DV6
00751 001057R        AND CBIT6       CLEAR 'DB' IN DV6
00752 000001X        STA $DV6,I      SAVE NEW STATUS
00753 000005X CON    LDA $DV18,I     GET STATUS WORD
00754 001052R        AND B5002       GET EOF,EOT & EOV BITS
00755 001044R        LDB BIT7        SET BIT 7
00756 002002         SZA             EOF, EOT, OR EOV?
00757 001036R        JSB DV6ER       YES, SET 'EOF' IN DV6
00760 000005X        LDA $DV18,I     GET STATUS WORD
00761 001050R        AND B2000       GET LOAD POINT BIT
00762 001043R        LDB BIT4        SET BIT 4
00763 002002         SZA             LOAD POINT?
00764 001036R        JSB DV6ER       YES, SET 'BOM' IN DV6
00765 000005X        LDA $DV18,I     GET STATUS WORD
00766 001047R        AND B1002       GET EOT & EOV BITS
00767 000406R        LDB B40         SET BIT 5
00770 002002         SZA             EOT OR EOV?
00771 001036R        JSB DV6ER       YES, SET 'EOM' IN DV6
00772 000023R        LDB B2          NR ERROR MESSAGE
00773 000005X        LDA $DV18,I     GET STATUS WORD
00774 002011         SLA,RSS         TAPE INSERTED?
00775 001014R        JMP ERR         NO, SET 'NR' DV16=2
00776 000461R        LDB B6          WP ERROR MESSAGE
00777 001270R        IOR =B177277
01000 002007         INA,SZA,RSS     WRITE PROT & WRITE ERR SET?
01001 001014R        JMP ERR         YES, SET 'WP' DV16=6
01002 000460R        LDB B5          PE ERROR MESSAGE
01003 000005X        LDA $DV18,I     GET STATUS WORD
01004 001045R        AND B444
01005 002002         SZA             WRITE ERR, RD ERR OR HARD ERR?
01006 001014R        JMP ERR         YES, SET 'PE' DV16=5
01007 000457R        LDB B4          ET ERROR MESSAGE
01010 000005X        LDA $DV18,I     GET STATUS WORD
01011 001046R        AND B1000       GET EOT BIT
01012 002003         SZA,RSS         EOT?
01013 006400         CLB             NO, SET DV16=0
01014 001256R ERR    LDA DVX12,I     GET 'S','U' OR 'F'
01015 002003         SZA,RSS         ZERO LENGTH READ?
01016 001024R        JMP ZLNRD       YES, CHECK STATUS BITS
01017 000411R        AND LBYTE       REMOVE LOW BYTE
01020 000232R        CPA S           SUCCESSFUL?
01021 001031R        JMP SUCCS       YES
01022 001056R        CPA U           USER INTERRUPT?
01023 001055R        LDB RTRY        YES, RESTART
```

```
01024 000005X ZLNRD LDA $DV18,I    GET STATUS
01025 001053R       AND B7467      MASK SFT ERR,WRT PROT,CMND EXECUTION
01026 001051R       CPA B4001      EOF, TAPE INSERTED SET?
01027 006400        CLB            YES, ZERO ERROR CODE
01030 000024R       CPA B3         EOV, TAPE INSERTED SET?
01031 006400  SUCCS CLB            YES, ZERO ERROR CODE
01032 006002        SZB            ANY ERRORS?
01033 000202R       JMP DONE       YES, DONE (B=ERROR CODE)
01034 001267R       LDA DVX21,I    GET RETURN ADDRESS
01035 000000        JMP A,I        RETURN
*
01036 000000  DV6ER NOP
01037 000001X       LDA $DV6,I     GET DEVICE STATUS
01040 000001        IOR B          ADD STATUS BIT
01041 000001X       STA $DV6,I     SAVE NEW STATUS
01042 001036R       JMP DV6ER,I    RETURN
*
01043 000020  BIT4  OCT 20         'BOM' BIT
01044 000200  BIT7  OCT 200        'EOF' BIT
01045 000444  B444  OCT 444        'WRITE ERR','RD ERR','HARD ERR' BITS
01046 001000  B1000 OCT 1000       'EOT' BIT
01047 001002  B1002 OCT 1002       'EOT','EOV' BITS
01050 002000  B2000 OCT 2000       'LOAD POINT' BIT
01051 004001  B4001 OCT 4001       'EOF','TI' BITS
01052 005002  B5002 OCT 5002       'EOF','EOT','EOV' BITS
01053 007467  B7467 OCT 7467       MASK 'SE','WP','CE' BITS
01054 007777  B7777 OCT 7777
01055 100077  RTRY  OCT 100077     DON'T DOWN/DON'T FLUSH, RESTART
01056 052400  U     OCT 52400      'U', USER INTERRUPT
01057 177677  CBIT6 OCT 177677     CLEAR BIT 6
*
*
* FLUSH PORT BUFFERS FOR MUX (FUNCTION CODE = 26) *
*
01060 000000  FPORT NOP
01061 001261R       LDA DVX15,I    SETUP SUBFUNCTION
01062 001070R       AND SUBFN      FLUSH PORT BUFFERS
01063 001071R       IOR B2603      FOR MUX.
01064 000002X       STA $DV15,I    SAVE IT
01065 002404        CLA,INA
01066 000003X       STA $DV16,I    1ST PARAMETER = 1
01067 001060R       JMP FPORT,I    RETURN
*
01070 170000  SUBFN OCT 170000     CLEAR SUBFUN & RQ
01071 002603  B2603 OCT 2603       CNTRL REQ. (FC=26B)
*
*
* LOCATE ABSOLUTE FILE IPRM1 (FUNCTION CODE = 27) *
*
01072 000003X ABSF  LDA $DV16,I    GET ABSOLUTE FILE
01073 002020        SSA            NEGATIVE FILE #?
```

```
01074 001130R       JMP DDCM2      YES, DEVICE COMPLETE
01075 001120R       LDB M257
01076 000000        ADB A
01077 006021        SSB,RSS        FILE > 256
01100 001130R       JMP DDCM2      YES, DEVICE COMPLETE
01101 002300        CCE            E=1 FOR DECIMAL FILE #
01102 000011X       JSB $CVT3      CONVERT FILE # TO ASCII
01103 001231R       LDA UN
01104 000012X       IOR $CVT+1
01105 001251R       STA DVX7,I     SAVE <uSPACE OR NUMBER>
01106 000012X       LDA $CVT+2
01107 001252R       STA DVX8,I     SAVE FILE NUMBER
01110 001220R       LDA P2
01111 001253R       STA DVX9,I     SAVE <p2>
01112 001217R       LDA C
01113 001254R       STA DVX10,I    SAVE <C>
01114 001117R       LDA M13
01115 000004X       STA $DV17,I    BUFFER LENGTH
01116 000515R       JMP SEND
*
01117 177763 M13    DEC -13
01120 177377 M257   DEC -257
*
*
01121 001260R DDCOM LDA DVX14,I    GET TOTAL XLOG (+CHARS)
01122 002004        INA            ROUNDOFF
01123 001100        ARS            CONVERT TO WORDS
01124 001255R       LDB DVX11,I    GET ORIGIONAL XLOG
01125 006020        SSB            WORDS?
01126 001260R       LDA DVX14,I    NO, SAVE CHAR'S
01127 000004X       STA $DV17,I    YES, SAVE WORDS
01130 000003X DDCM2 LDA $DV16      ADDR OF INFO
01131 001262R       LDB DVX16      ADDR TO SAVE IT
01132 000016X       JSB .MVW       SAVE $DV16, $DV17, $DV18 & $DV19
01133 000457R        DEF B4          IN EXTENSION
01134 000000         NOP
01135 001241R       LDA ESCB       GET <ESCb> UNLOCK KEYBOARD
01136 001246R       STA DVX4,I     SAVE IT
01137 000642R       JSB ASCWT      ASCII WRITE (SYSTEM ADDR. SPACE)
01140 001246R       LDA DVX4       ESCAPE SEQUENCE ADDRESS
01141 000003X       STA $DV16,I    SAVE IT
01142 000231R       LDA M2         BUFFER LENGTH
01143 000004X       STA $DV17,I    SAVE IT
01144 002404        CLA,INA        ALLOW TIMEOUT
01145 001155R       JSB CEXIT      INITIATE UNLOCK KEYBOARD
01146 001262R       LDA DVX16      ADDR OF INFO
01147 000003X       LDB $DV16      ADDR TO RESTORE IT
01150 000016X       JSB .MVW       RESTORE $DV16, $DV17, $DV18 & $DV19
01151 000457R        DEF B4          FROM EXTENSION
01152 000000         NOP
*
```

```
01153 002400        CLA
01154 000000R       JMP DD.20,I    DEVICE COMPLETE
*
*
* CONTINUATION EXIT *
*
01155 000000  CEXIT NOP
01156 001155R       LDB CEXIT      STORE RETURN ADDR
01157 001243R       STB DVX1,I     AT DVX1
01160 000003X       LDB $DV16,I    GET BUFFER ADDR
01161 001266R       STB DVX20,I    SAVE ADDR OF CURRENT READ
01162 000000R       ISZ DD.20
01163 000000R       JMP DD.20,I    INTERFACE INITIATE
*
*
* CONTINUATION *
*
01164 000003X CONT  LDA $DV16,I    GET ERROR CODE
01165 000467R       AND B77
01166 001246R       LDB DVX4,I     KEYBOARD JUST
01167 001241R       CPB ESCB       UNLOCKED?
01170 001173R       JMP CONT2      YES, DEVICE COMPLETE
01171 002002        SZA            ANY ERRORS?
01172 001130R       JMP DDCM2      YES, DEVICE COMPLETE
01173 001243R CONT2 LDB DVX1,I
01174 000001        JMP B,I        CONTINUE REQUEST
*
*
* ROUTINE FOR DEFINING STORAGE IN DEVICE DVR EXT. *
*
01175 000000  SETAD NOP
01176 000007X       LDA $DV22,I   GET ADDR POINTING TO ADDR OF DVT EXT
01177 001243R       CPA DVX1      EXTENSION SETUP?
01200 001175R       JMP SETAD,I   YES, RETURN
01201 001212R       LDB D.21      SET FOR 21 MISC. STORAGE
01202 001213R       STB TEMP
01203 001242R       LDB DVX       SETUP
01204 000001        STA B,I       DVX1-DVX21
01205 002004        INA           ADDRESS
01206 006004        INB           POINTERS
01207 001213R       ISZ TEMP
01210 001204R       JMP *-4
01211 001175R       JMP SETAD,I   RETURN
*
01212 177753  D.21  DEC -21
01213 000000  TEMP  NOP           TEMPORARY STORAGE
*
*
*       REWIND/WRITE EOF/WRITE EOD
* ESC&  **************************
* P1(P2)
* U0(U5)(U6)
* C
*
01214 015446  ESC&   OCT 15446     <ESC&>
01215 070061  P1     OCT 70061     <p1>
01216 072460  U0     OCT 72460     <u0>
01217 041400  C      OCT 41400     <C>
*
01220 070062  P2     OCT 70062     <p2>
01221 072465  U5     OCT 72465     <u5>
01222 072466  U6     OCT 72466     <u6>
*
*       FORWARD/BACKWARD SPACE 1 RECORD/FILE
* ESC&  ************************************
* P1(P2)
* UFRWD(UBKWD)
* ONEP OR TWOP
* ONEC(TWOC)
*
01223 072453  UFRWD  OCT 72453     <u+>
01224 030560  ONEP   OCT 30560     <1p>
01225 031160  TWOP   OCT 31160     <2p>
01226 030503  ONEC   OCT 30503     <1C>
*
01227 072455  UBKWD  OCT 72455     <u->
01230 031103  TWOC   OCT 31103     <2C>
*
*       FIND THE NTH FILE ON CTU (1 OR 2)
* ESC&  *********************************
* P1(P2)
* UN
* P2
* C
*
01231 072400  UN     OCT 72400     <u >
*
*       WRITE N BYTES TO CTU (1 OR 2)
* ESC&  *****************************
* P1(P2)
* DN
* W
*
01232 062000  DN     OCT 62000     <d >
01233 053400  W      OCT 53400     <W>
01234 002400  ENQ    OCT 2400      <ENQ>
*
*       READ FROM CTU (1 OR 2) TO COMPUTER
* ESC&  **********************************
* P1(P2)
* S2
* R
```

```
*
01235 071462  S2    OCT 71462     <s2>
01236 051000  R     OCT 51000     <R>
*
*       FETCH STATUS OF CTU (1 OR 2)
* ESC&  ****************************
* P1(P2)
* UP
*
01237 057000  UP    OCT 57000     <^>
*
*       LOCK/UNLOCK KEYBOARD
* ESCC  ********************
* ESCB
*
01240 015543  ESCC  OCT 15543     <ESCc>
01241 015542  ESCB  OCT 15542     <ESCb>
*
* EXTENSION FOR MISC. STORAGE *
*
01242 001243R DVX   DEF DVX1
01243 000000  DVX1  NOP           CONTINUATION ADDR
01244 000000  DVX2  NOP           BUFF ADDR OF CURRENT REQUEST
01245 000000  DVX3  NOP           BUFF LENGTH (-CHAR'S)
01246 000000  DVX4  NOP           ESCc OR ESCb
01247 000000  DVX5  NOP           ESC&
01250 000000  DVX6  NOP           P1(P2)
01251 000000  DVX7  NOP             REMAINING
01252 000000  DVX8  NOP             CONTROL
01253 000000  DVX9  NOP             ESCAPE
01254 000000  DVX10 NOP             SEQUENCE
01255 000000  DVX11 NOP           INITIAL LENGTH
01256 000000  DVX12 NOP           ADDRESS OF 'S' OR 'F'
01257 000000  DVX13 NOP           REQUEST LENGTH (+CHARS)
01260 000000  DVX14 NOP           CHARACTER ACCUMULATOR
01261 000000  DVX15 NOP           INITIAL SUBFUNCTION
01262 000000  DVX16 NOP           BUFFER ADDR
01263 000000  DVX17 NOP             FOR
01264 000000  DVX18 NOP             1-8
01265 000000  DVX19 NOP             BYTE READ
01266 000000  DVX20 NOP           ADDR OF CURRENT READ
01267 000000  DVX21 NOP           CONTINUATION ADDR FOR DYNAMIC STATUS
*
* DRIVER PARAMETER STORAGE *
*
*     $DVTP    CTU LEFT OR RIGHT
*
*
01270 177277
                    END
```

```
Macro/1000 Cross reference

* - Volatile reference (store, jump, call...)

$CVT . . . . . . . . . 21:   127    129    704    706
$CVT3  . . . . . . . . 21:   125*   702*
$DV1 . . . . . . . . . 21:   318    328
$DV15  . . . . . . . . 20:    80    146*   166*   200*   245*   260*   296
                             320    526*   554    556*   683*
$DV16  . . . . . . . . 20:    94     97*   148*   168*   188*   193*   247*
                             268*   379*   419*   515*   550*   558*   685*
                             694    727    736*   742    756    765
$DV17  . . . . . . . . 20:    98    134*   150*   170*   176    195*   233*
                             249*   256    287*   298    417*   513*   548*
                             560*   713*   726*   738*
$DV18  . . . . . . . . 20:   152*   172*   202*   251*   266*   528*   564*
                             579*   588    593    597    604    609    614
                             620    628    633    645
$DV19  . . . . . . . . 20:   142*   204*   241*   292*   529*   562*
$DV22  . . . . . . . . 20:   779
$DV6 . . . . . . . . . 20:    87     89*   400    442    446    460    464
                             568    570*   600    603*   657    659*
$DVTP  . . . . . . . . 21:    77
$ONER  . . . . . . . . 21:   316*
$ONEW  . . . . . . . . 21:   326*
.MVW . . . . . . . . . 21:   729*   743*
A  . . . . . . . . . . 33:   120    654*   698
ABSF . . . . . . . . .694:   376*
ACK  . . . . . . . . .219:   159
ASCWT  . . . . . . . .521:    82*   450*   468*   476*   486*   530*   544*
                             734*
B  . . . . . . . . . . 34:   279*   291    658    773*   785*
B1 . . . . . . . . . . 62:    53    105    257    355    409
B10  . . . . . . . . .385:   367    587    589
B1000  . . . . . . . .665:   634
B1002  . . . . . . . .666:   615
B13  . . . . . . . . .386:   369    507
B14  . . . . . . . . .387:   371    488    503    509
B1415  . . . . . . . .334:   250
B2 . . . . . . . . . . 63:    55    118    357    452    501    601    619
B200 . . . . . . . . .391:   474
B2000  . . . . . . . .667:   610
B26  . . . . . . . . .388:   373    411
B2603  . . . . . . . .689:   682
B27  . . . . . . . . .389:   375
B3 . . . . . . . . . . 64:    57     84     85     93    144    225    359
                             469    478    555    649
B320 . . . . . . . . .392:   401
B4 . . . . . . . . . .382:    48    361    583    632    730    744
B40  . . . . . . . . .332:   324    616
B4001  . . . . . . . .668:   647
```

```
B444 . . . . . . . . .664:   629
B5 . . . . . . . . . .383:   363     627
B5002  . . . . . . . .669:   605
B6 . . . . . . . . . .384:   365     623
B602 . . . . . . . . .536:   524
B7 . . . . . . . . . . 65:    41
B7467  . . . . . . . .670:   646
B77  . . . . . . . . .390:   352     766
B7700  . . . . . . . .393:   473
B7777  . . . . . . . .671:   578
BIT4 . . . . . . . . .662:   443     461     594     611
BIT6 . . . . . . . . .209:   243     584     592
BIT7 . . . . . . . . .663:   447     465     606
BIT8 . . . . . . . . .210:   165
BS2R . . . . . . . . .450:   496*
BSRF . . . . . . . . .434:   358*    372*
C  . . . . . . . . . .805:   414     710
CBIT6  . . . . . . . .674:   602
CBIT7  . . . . . . . .211:   164
CEXIT  . . . . . . . .753:    49*    136*    139*    154*    174*    179*    235*
                             238*    253*    294*    421*    424*    436*    439*
                             454*    457*    480*    483*    490*    493*    552*
                             566*    740*    754
CNTRL  . . . . . . . .350:    86*
CON  . . . . . . . . .604:   599*
CONT . . . . . . . . .765:    56*
CONT2  . . . . . . . .772:   769*
CR.CR  . . . . . . . .218:   264
D.21 . . . . . . . . .792:   782
DC1  . . . . . . . . .333:   203     240     290     561
DD.20  . . . . . . . . 37:   748*    758*    759*
DDCM2  . . . . . . . .727:    50*     58*    380*    696*    700*    771*
DDCOM  . . . . . . . .720:    59*    189*    314*    329*
DIREC  . . . . . . . . 68:    38*     40
DN . . . . . . . . . .841:   126
DONE . . . . . . . . .188:   184*    345*    426*    445*    463*    467*    475*
                             652*
DV6ER  . . . . . . . .656:   586*    591*    596*    608*    613*    618*    660*
DVX  . . . . . . . . .870:   784
DVX1 . . . . . . . . .871:   755*    772     780     870
DVX10  . . . . . . . .880:   711*
DVX11  . . . . . . . .881:    99*    723
DVX12  . . . . . . . .882:   181     192     343*    398*    637
DVX13  . . . . . . . .883:   282*    301
DVX14  . . . . . . . .884:    47*    177*    185*    310*    342*    720     725
DVX15  . . . . . . . .885:    81*     83      90      92*    113     143     163
                             196     242     259     317     327     350     472
                             522     680
DVX16  . . . . . . . .886:   147     157     246     261     269     557     728
                             741
DVX17  . . . . . . . .887:   270
```

```
DVX18  . . . . . . . .888:  572
DVX19  . . . . . . . .889:  573
DVX2 . . . . . . . . .872:   95*   167    267
DVX20  . . . . . . . .890:  312    757*
DVX21  . . . . . . . .891:  543*   653
DVX3 . . . . . . . . .873:  104*   111    169    283    286    303
DVX4 . . . . . . . . .874:   74*    96    418    514    549    733*   735
                            767
DVX5 . . . . . . . . .875:   76*
DVX6 . . . . . . . . .876:   79*
DVX7 . . . . . . . . .877:  128*   229*   413*   505*   546*   705*
DVX8 . . . . . . . . .878:  130*   231*   415*   499*   707*
DVX9 . . . . . . . . .879:  132*   511*   709*
DYST . . . . . . . . .541:  186*   344*   399*   425*   440*   458*   484*
                            494*   542
ECHO . . . . . . . . .212:   91
ENQ  . . . . . . . . .843:  141
EOF  . . . . . . . . .397:  368*
ERR  . . . . . . . . .637:  622*   626*   631*
ERROR  . . . . . . . .378:  123*
ESC& . . . . . . . . .802:   75
ESCB . . . . . . . . .866:  732    768
ESCC . . . . . . . . .865:   73
FBRF . . . . . . . . .498:  431*   435*   453*   479*   489*   517*
FPORT  . . . . . . . .679:  138*   237*   686*
FSRF . . . . . . . . .430:  227*   360*   370*   470*
GO . . . . . . . . . . 53:   43*
ILREQ  . . . . . . . .216:  378
INIT . . . . . . . . . 73:   54*
LBYTE  . . . . . . . .335:   88    158    182    319    569    640
M1 . . . . . . . . . .331:  302    305
M11  . . . . . . . . . 67:  133
M12  . . . . . . . . .535:  512
M13  . . . . . . . . .716:  712
M2 . . . . . . . . . .214:  194    307    737
M257 . . . . . . . . .717:  119    697
M5 . . . . . . . . . .532:  248
M7 . . . . . . . . . .533:  547
M8 . . . . . . . . . .534:  559
M9 . . . . . . . . . . 66:  232    416
ONEC . . . . . . . . .821:  506
ONEP . . . . . . . . .819:  430    434    477    487
P1 . . . . . . . . . .803: Symbol not referenced
P2 . . . . . . . . . .807:  708
PLU  . . . . . . . . .213:   78
R  . . . . . . . . . .852:  230
RACK . . . . . . . . .142:  161*
READ . . . . . . . . .225:  106*
READ5  . . . . . . . .241:  258*
RECFL  . . . . . . . .472:  449*
RQASC  . . . . . . . .537:  197    523
```

```
RS.CR  . . . . . . . .217:   262
RTRY . . . . . . . . .672:   644
RW . . . . . . . . . .408:   354*    356*    362*    364*    374*
S  . . . . . . . . . .215:   183     340     397     641
S2 . . . . . . . . . .851:   228
SBIT . . . . . . . . .220:   145     199     244     525
SEND . . . . . . . . .418:   714*
SEND1  . . . . . . . .421:   432*
SETAD  . . . . . . . .778:    39*    781*    790*
STAT . . . . . . . . .191:   178*    206*    423*    438*    456*    482*    492*
SUBFN  . . . . . . . .688:   681
SUCCS  . . . . . . . .650:   642*
TEMP . . . . . . . . .793:   315*    325     783*    788*
TICST  . . . . . . . .340:   366*
TWOC . . . . . . . . .824:   508     510
TWOP . . . . . . . . .820:   451
U  . . . . . . . . . .673:   643
U0 . . . . . . . . . .804:   408
U5 . . . . . . . . . .808:   410
U6 . . . . . . . . . .809:   412
UBKWD  . . . . . . . .823:   502     504
UFRWD  . . . . . . . .818:   500
UN . . . . . . . . . .833:   703
UP . . . . . . . . . .859:   545
W  . . . . . . . . . .842:   131
WRITE  . . . . . . . .111:  Symbol not referenced
XLOG . . . . . . . . .308:   300*
ZEROL  . . . . . . . .342:   263*    265*    278*
ZERR . . . . . . . . .377:   403*
ZLNRD  . . . . . . . .645:   639*
Macro:   No   errors total
```

Chapter 4
Interface Driver

The system enters the interface driver as indicated below. The address of the driver is picked up from the IFT.

All pointers to the IFT, as described in the chapter on System I/O Tables, are set prior to entering the driver. The registers and calling sequence are (global register = select code and global register enabled):

```
    B-register = DVT Address

    A-register: Bits 2-0 = Entry Directive, as below:

    JSB ID.nn

    P+1  done

    P+2  wait

    P+3  resume
```

Although not normally needed, the driver can determine the select code for the interface card by an LIA 2 instruction.

The various entry directives and their codes in the A-register are:

Code	Meaning
000	Abort
001	Initiate
010	Continue
011	Time Out
100	Power Fail

The driver must increment its return address, stored at its entry point, to the proper exit as follows:

Source Code	Meaning	A-register on Exit
JMP ID.XX,I P+1 return	Request complete on interface driver.	Q D 0 H T
ISZ ID.XX JMP ID.XX,I P+2 return	Wait for next interrupt or time-out.	0 0 I H T
ISZ ID.XX ISZ ID.XX JMP ID.XX,I P+3 return	Resume processing in the device driver. An interrupt has occurred from a device whose driver is not at the request list head (IFT3).	0 0 0 H T

The P+3 return from the interface driver essentially means that the interface driver does not have enough information to completely process the interrupt. Therefore, it must call upon the device driver.

Upon exit from the driver bits 0-4 of the A-register are stored in the system flags area of IFT7. The meanings of the bits are:

Q = Do not advance to next request on list.

D = Defer entering device driver (pseudo done).

I = Report illegal interrupt.

H = Assert or maintain hold on new request initiation.

T = Set time-out on device request.

Entry Directives

The system will set up the pointers to the IFT before entering the driver. The system will also set up the pointers to the DVT if this is an "initiate" or "abort" entry. For other entry directives, the driver may set up pointers to the DVT by calling system routine $DIOC (as required).

Upon entry, the directive code will be in the A-register bits 2-0 and the DVT address will be in the B-register.

The global register for the select code given in the IFT is enabled prior to the entry of the interface driver by the system. The select code (if needed) can be found by reading the global register (LIA 2 instruction).

Initiate New Request

Upon entry, bits 2-0 of the A-register equal 001.

The purpose of this directive is to start a new request. The request code is in DVT15 with parameters in DVT16 through DVT19.

The driver parameter area (starting at DVTP) and the driver communication area of DVT20 may also contain useful information for processing the request.

Unless the interface driver can complete the request immediately, it should make a "wait" exit after initializing the I/O operation. It should expect a "continue" entry to process the next interrupt, which will normally be a DMA completion.

Continue Processing

Upon entry, bits 2-0 of the A-register equal 010.

The purpose of this directive is to handle an interrupt, which usually will indicate DMA completion. The driver might chose to issue a new command which would lead to another interrupt or complete the request and take the "done" exit.

Upon receiving this directive, the driver should immediately test and clear both flag 30 and flag 23. The system itself takes no action on the flags.

1. Flag 30 is the interface card flag and is cleared with a CLF 30. Either the interface flag or the DMA flag 21 may be used to indicate completion.

2. Flag 23 is set if one or more of the following flags are set:

   Flag 20, indicating end of DMA chained list.
   Flag 21, indicating DMA completion.
   Flag 22, indicating a DMA parity error.

   A CLF 23 will clear flags 20, 21, and 22.

Timeout Processing

Upon entry, bits 2-0 of the A-register equal 011.

The interface driver is called for time-out when the working clock in IFT2 is incremented to zero.

The working clock is initialized by the system upon every entry to the interface driver. It is set to the value taken from DVT13.

The clock is used only if the interface driver sets the T bit in the A-register upon exit. If it is enabled by this bit, then the clock starts ticking upon exit from the interface driver.

Abort Request

Upon entry, bits 2-0 of the A-register equal 000.

Prior to entering the driver (device or interface) the "A" bit in DVT7 is set to indicate that abort processing is in progress. It will be reset when abort processing is completed by the drivers.

For requests which are busy, the device driver is given first chance at abort processing. If the device driver is entered and handles the request, then the interface driver will be called for abort processing only if the device driver makes an "initiate exit" with the abort request in the A-register.

If the user request specifies that the device driver is bypassed (bit 15 in the control word) or no device driver exists, then the system initiates the abort request on the interface driver. The system will also initiate this request if the device driver treats the abort request as an "illegal request."

The intent of the abort request is to stop the operation on the I/O card as soon as possible. This may result in unpredictable device action. Therefore it is best if the action is initiated only upon the decision of the device driver. In any case, it is the responsibility of the interface driver to return the I/O card to a known state after completing the abort.

When done with abort processing, the driver should take the "done" exit.

Power-Fail Restart

Upon entry, bits 2-0 of the A-register equal 100.

The interface driver will always be called upon power-fail restart. Hence every interface driver must be coded to accept such an entry directive (although it may choose to ignore it).

The interface driver will always be called prior to the device driver when power-fail processing is to be done. The device driver will be called after the interface driver only if the P bit is set in DVT4.

Driver Exit

Upon driver exit, there are three concerns:

1. Setting of system flags through bits in the A-register.

2. Posting status in the DVT.

3. Posting any errors, in addition to status.

The system flags are set regardless of whether the exit is to indicate "done," "wait" or "resume". However, status and errors are posted only on the done exit.

It is important to remember that the status of the transfer of data and any tranfer errors should be posted by the interface driver. The device driver handles only device-dependent status and errors.

The topics of status and error posting are common to both the device driver and the interface driver and so they are covered in a separate chapter of this manual.

System Flags

The three possible exit sequences from the interface driver are given below. For each exit, bits 4-0 of the A-register have the meaning indicated. The B-register is meaningless.

The system takes the contents of A-register bits 4 through 0 and places them in the system flags area of IFT7.

A-register Bit:	4	3	2	1	0
P + 1 "Done"	Q	D	0	H	T
P + 2 "Wait"	0	0	I	H	T
P + 3 "Resume"	0	0	0	H	T

T means set time-out. If set, the system will enter the interface driver in the time-out list. See Time-out Processing.

H means hold. If set, the system will delay calling the interface driver to start a new request. It is recommended that the driver set this bit when it exits with DMA active for the user's buffer. This prevents the DMA port map register from being altered while DMA is in progress.

If the hold is made on a "done" exit, a "continue" entry will be made to the device driver, just as if the hold was not made. This puts the IFT in a non-busy state in which the driver is waiting for expected interrupts. For example, the driver might be waiting for a response to a serial poll on the HPIB. When it comes, the interrupt causes a "continue" entry. The driver can easily identify the reason for the entry because the IFT is not busy.

I indicates an illegal interrupt. If set, the system will issue an error message of the form:

> Illegal interrupt from SC nn
>
> where yy is the select code on which the interrupt occurred.

Q is request advance inhibit. If set, then the current DVT remains at the head of the request list on IFT3. Requests linked on other DVTs will be held off. (See Figure 2-1 for more on DVT/IFT linking.) Note that, even if the Q bit is zero, the request will remain at the head of the list if the IFT is locked to the DVT.

D defers entry to device driver. If set, then the continue entry to the device driver will not be made; hence the request completion will be delayed. This is a "pseudo done" exit.

> If the driver sets the D bit, then it must keep track of the request and complete it later, if needed. The action taken by the system is simply to avoid the continue entry into the device driver.
>
> Normally, if D is set, then Q is not set, permitting advance to the next request. Thus, requests from multiple devices may be made on the interface driver before any are completed. This may be valuable if the requests take a long time to complete.
>
> The use of this bit implies timeout control by the interface driver. See the section on Asychronous I/O and Polling.

Sample Interface Driver

This section contains a listing for a sample interface driver. Many of the features of the driver are not explained in detail in the manual because they are not essential to the structure of the driver. That is, there are many different ways the same result could be achieved and this listing represents one programmer's approach.

Although this sample driver has been tested, it is not guaranteed to correspond to the code in any driver shipped with the system. It is included here only as an example.

```
             ASMB,q,c
*
*     NAME:   ID.00
*     SOURCE: 92071-18082
*     RELOC:  92071-16082
*     PGMR:   T.A.L.
*
*   ***************************************************************
*   * (C) COPYRIGHT HEWLETT-PACKARD COMPANY 1980.  ALL RIGHTS      *
*   * RESERVED.  NO PART OF THIS PROGRAM MAY BE PHOTOCOPIED,       *
*   * REPRODUCED OR TRANSLATED TO ANOTHER PROGRAM LANGUAGE WITHOUT*
*   * THE PRIOR WRITTEN CONSENT OF HEWLETT-PACKARD COMPANY.        *
*   ***************************************************************
*
                   NAM ID.00,0     92071-16082  REV.2326 <830519.1303>
*
                   ENT ID.00
                   EXT $IFTX,$DV15,$DV16,$DV17,$DV18,$DV19
                   EXT $IF1,$IF5,$IF6,$DIOC,$LUTA,$DMPR,$SELR,$DVTP
*
                   GEN 1,PA
                   GEN 7,EID.00,TX:33
*
      000000  A    EQU 0
      000001  B    EQU 1
*
00000 000000  ID.00 NOP
00001 000001X      LDB $IFTX      GET INTERFACE DRIVER STORAGE ADDR
00002 000667R      STB DMAAD      SAVE IT
00003 000110R      ADB D13        COMPUTE BREAK FLAG ADDR
00004 000670R      STB BRKFL      SAVE IT
00005 006004       INB            COMPUTE PARITY CHECK FLAG ADDR
00006 000671R      STB PCHKB      SAVE IT
00007 006004       INB            COMPUTE IGNORE INPUT FLAG ADDR
```

```
00010 000672R       STB IGNOR     SAVE IT
00011 006004        INB           COMPUTE BIT BUCKET ADDR
00012 000673R       STB BITBK     SAVE IT
*
* USE 33 WDS ONLY TO STAY COMPATABLE WITH ID.01     (A.83BC)
* Use .setp in A systems
00013 006004        INB                                              *BC*
00014 000674R       STB WD18A     MODEM STATUS WORD ADDRESS          *BC*
00015 006004        INB                                              *BC*
00016 000675R       STB WD19A     RETRY RESUME ADDRESS               *BC*
00017 006004        INB                                              *BC*
00020 000676R       STB WD20A     \                                  *BC*
00021 006004        INB            \                                 *BC*
00022 000677R       STB WD21A       > MODEM ALARM PROGRAM NAME       *BC*
00023 006004        INB            /                                 *BC*
00024 000700R       STB WD22A     /                                  *BC*
00025 006004        INB                                              *BC*
00026 000701R       STB WD23A     LOGLU FOR ALARM PROG               *BC*
00027 006004        INB                                              *BC*
00030 000702R       STB SAVEA     TEMP A-REG STORAGE                 *BC*
00031 006004        INB                                              *BC*
00032 000703R       STB STSSS     CARD STATUS SNAP SHOT              *BC*
00033 006004        INB                                              *BC*
00034 000704R       STB RQ        REQUEST WORD                       *BC*
00035 006004        INB                                              *BC*
00036 000705R       STB W18                                          *BC*
00037 006004        INB                                              *BC*
00040 000706R       STB DIREC     ENTRY DIRECTIVE                    *BC*
00041 006004        INB                                              *BC*
00042 000707R       STB STRA      TEMP STORAGE                       *BC*
00043 006004        INB                                              *BC*
00044 000710R       STB STRB       "       "                         *BC*
*
00045 000112R       STA T         SAVE ENTRY DIRECTIVE             A.83BC
00046 006004        INB           IFTX WD31                        A.83BC
00047 000711R       STB CRLFA     MOVE CRLF CODE TO EXTENSION      A.83BC
00050 000630R       LDA CRLFX     IN CASE DRIVER GETS MAPPED OUT   A.83BC
00051 000001        STA B,I                                        A.83BC
*
00052 006004        INB           IFTX WD32                        A.83BC
00053 000712R       STB ESCA      MOVE ESCA CODE TO EXTENTION      A.83BC
00054 000631R       LDA ESCX      IN CASE DRIVER GETS MAPPED OUT   A.83BC
00055 000001        STA B,I                                        A.83BC
*
00056 006004        INB           IFTX WD33                        A.83BC
00057 000713R       STB DC1A      MOVE DC1 CODE TO EXTENTION       A.83BC
00060 000632R       LDA DC1X      IN CASE DRIVER GETS MAPPED OUT   A.83BC
00061 000001        STA B,I                                        A.83BC
*
00062 000112R       LDA T         RETRIEVE ENTRY DIRECTIVE         A.83BC
00063 000106R       AND B7
```

```
00064 000103R        LDB ABRTE     DON'T DOWN/DO FLUSH, NO MESSAGE
00065 002003         SZA,RSS       ABORT?
00066 000615R        JMP ABORT     YES
*
00067 000104R        CPA B1        INITIATE?
00070 000113R        JMP INIT      YES
00071 000706R        STA DIREC     SAVE DIRECTIVE
00072 002404         CLA,INA
00073 000010X        LDB $IF5,I    GET DVT ADDRESS
00074 000012X        JSB $DIOC     SET UP DVT POINTERS
00075 000706R        LDA DIREC     RESTORE DIRECTIVE
00076 000105R        CPA B2        CONTINUATION?
00077 000442R        JMP CONT      YES
00100 000606R        CPA B3        TIMEOUT?
00101 000614R        JMP TIMOT     YES
*
* POWERFAIL *
*
00102 000447R        JMP PWRFL     POWERFAIL
*
00103 140077  ABRTE OCT 140077    ABORT ERROR CODE
00104 000001  B1    OCT 1
00105 000002  B2    OCT 2
00106 000007  B7    OCT 7
00107 003000  B3000 OCT 3000
00110 000015  D13   DEC 13
00111 000000  ZERO  NOP
00112 000000  T     NOP
*
*
* INITIATION *
*
00113 002400  INIT  CLA
00114 000672R       STA IGNOR,I   ZERO IGNORE INPUT FLAG
00115 000002X       LDA $DV15,I   GET RQ
00116 000606R       AND B3
00117 000606R       CPA B3        CONTROL REQUEST?
00120 000367R       JMP CNTRL     YES
*
* BUILD DEFAULT CONTROL WORD *
*
00121 000005X       LDA $DV18,I   GET USER CONTROL WORD
00122 000714R       AND =B174377  ZERO XMIT,RCV & CHLN BITS
00123 000002X       LDB $DV15,I   GET SUBFUNCTION & RQ
00124 000362R       IOR B1000     SET XMIT, BIT 9
00125 004032        SLB,RBL       WRITE REQUEST?
00126 000107R       XOR B3000     NO, SET RCV, BIT 10
00127 005727        BLF,BLF       SHIFT BINARY-ASCII BIT
00130 006020        SSB           BINARY?
00131 000140R       JMP SET8      YES
00132 000016X       LDB $DVTP,I   GET TERMINAL CONFIGURATION WORD
```

```
00133 005200        RBL            GET ASCII BIT
00134 006020        SSB            8 BIT ASCII ENABLED?
00135 000140R       JMP SET8       YES
00136 000671R       XOR PCHKB,I    7 BIT ASCII, ADD ERROR CHECKING
00137 000141R       JMP SET7       NO
00140 000361R SET8  IOR BIT8       YES, SET FOR 8 DATA BITS
*
00141 000005X SET7  STA $DV18,I    SAVE CARD CONTROL WORD
00142 000002X       LDA $DV15,I    GET RQ
00143 000004X       LDB $DV17,I    GET TRANSMISSION LOG
00144 000010        SLA            READ REQUEST?
00145 000265R       JMP READ       YES
*
*
* WRITE REQUEST *
*
00146 000002X       LDA $DV15,I    CHECK FOR BINARY/ASCII
00147 000357R       AND BIT6
00150 002003        SZA,RSS        ASCII?
00151 000163R       JMP ASCII      YES
*
00152 006003  BINRY SZB,RSS        BINARY ZERO XLOG?
00153 000611R       JMP ZLOG       YES, INTERFACE COMPLETE
00154 000633R       JSB QUAD       BUILD DATA QUAD, NO 'CRLF'
00155 100002X       DEF $DV15,I
00156 071400        OCT 71400      DMA CONTROL WORD
00157 100005X       DEF $DV18,I    CARD CONTROL WORD
00160 100003X       DEF $DV16,I    BUFFER ADDRESS
00161 100004X       DEF $DV17,I    BUFFER LENGTH
00162 000243R       JMP ID.IO      SEND DATA
*
00163 000002X ASCII LDA $DV15,I    GET SUBFUNCTION
00164 000363R       AND BIT11      GET ESC BACKARROW BIT
00165 002002        SZA            PERFORM ESC BACKARROW?
00166 000212R       JMP ASBLK      YES
*
* CHARACTER MODE *
*
00167 000002X       LDA $DV15,I    CHECK FOR 'CRLF'
00170 000360R       AND BIT7
00171 002002        SZA            ADD CRLF?
00172 000152R       JMP BINRY      NO
00173 006003        SZB,RSS        ASCII ZERO XLOG?
00174 000203R       JMP CRLF1      YES
00175 000633R       JSB QUAD       BUILD DATA QUAD
00176 100002X       DEF $DV15,I
00177 171400        OCT 171400     DMA CONTROL WORD
00200 100005X       DEF $DV18,I    ASIC CONTROL WORD
00201 100003X       DEF $DV16,I    BUFFER ADDRESS
00202 100004X       DEF $DV17,I    BUFFER LENGTH
*
```

```
00203 000633R CRLF1 JSB QUAD      BUILD 'CRLF' QUAD
00204 000111R       DEF ZERO
00205 071400        OCT 71400     DMA CONTROL WORD
00206 100005X       DEF $DV18,I   ASIC CONTROL WORD
00207 000711R       DEF CRLFA     CRLF ADDRESS
00210 000354R       DEF M2        BUFFER LENGTH
00211 000243R       JMP ID.IO     SEND DATA
*
* BLOCK MODE *
*
00212 006003  ASBLK SZB,RSS       ASCII ZERO XLOG?
00213 000222R       JMP CRLFQ     YES, OUTPUT CRLF
00214 000633R       JSB QUAD      BUILD DATA QUAD
00215 100002X       DEF $DV15,I
00216 171400        OCT 171400    DMA CONTROL WORD
00217 100005X       DEF $DV18,I   ASIC CONTROL WORD
00220 100003X       DEF $DV16,I   BUFFER ADDRESS
00221 100004X       DEF $DV17,I   BUFFER LENGTH
*
00222 000002X CRLFQ LDA $DV15,I   CHECK FOR 'CRLF'
00223 000360R       AND BIT7
00224 002002        SZA           ADD CRLF?
00225 000235R       JMP NOCR      NO
*
00226 000633R       JSB QUAD      BUILD 'CRLF ESC DC1' QUAD
00227 000111R       DEF ZERO
00230 071400        OCT 71400     DMA CONTROL WORD
00231 100005X       DEF $DV18,I   ASIC CONTROL WORD
00232 000711R       DEF CRLFA     CRLF ADDRESS
00233 000356R       DEF M5        BUFFER LENGTH
00234 000243R       JMP ID.IO     SEND DATA
*
00235 000633R NOCR  JSB QUAD      BUILD 'ESC DC1' QUAD
00236 000111R       DEF ZERO
00237 071400        OCT 71400     DMA CONTROL WORD
00240 100005X       DEF $DV18,I   ASIC CONTROL WORD
00241 000712R       DEF ESCA      ESC ADDRESS
00242 000355R       DEF M3        BUFFER LENGTH
*
*
* START DMA *
*
00243 002404  ID.IO CLA,INA       ALLOW TIMEOUT
00244 000000R       ISZ ID.00     SETUP FOR INTERFACE CONTINUE
00245 107721  WDOUT CLC 21B,C     SUSPEND AND
00246 107723        CLC 23B,C     TERMINATE DMA OPERATION
00247 006400        CLB
00250 106624        OTB 24B       CLEAR BREAK FLAG
00251 106631        OTB 31B       CLEAR XMIT & RCV
00252 103730        STC 30B,C     ENABLE BREAK
*
```

```
00253 000001X       LDB $IFTX     GET QUAD
00254 006004        INB           STARTING ADDRESS
00255 106620        OTB 20B
00256 103720        STC 20B,C     START DMA
00257 000000R       JMP ID.00,I   INTERFACE COMPLETE/CONTINUE
*
00260 103730  WAIT  STC 30B,C
00261 002404        CLA,INA       ALLOW TIMEOUT
00262 000670R       STA BRKFL,I   SET BREAK FLAG
00263 000000R       ISZ ID.00
00264 000000R       JMP ID.00,I   INTERFACE CONTINUE
*
*
* READ REQUEST *
*
00265 006003  READ  SZB,RSS       ZERO XLOG?
00266 000611R       JMP ZLOG      YES, INTERFACE COMPLETE
00267 000006X       LDA $DV19,I   GET OPTIONAL PARAMETER
00270 000365R       AND LBYTE     REMOVE LOWER BYTE
00271 002003        SZA,RSS       HIBYTE > 0?
00272 000305R       JMP READB     NO, CHECK LOW BYTE
00273 000005X       LDA $DV18,I   GET ASIC CONTROL WORD
00274 000364R       AND M1011     REMOVE RCV & ECHO BITS
00275 000362R       IOR B1000     SET XMIT BIT
00276 000627R       STA TEMP      SAVE ASIC CONTROL WORD
00277 000633R       JSB QUAD      BUILD WRITE QUAD
00300 000111R       DEF ZERO
00301 171400        OCT 171400    DMA CONTROL WORD
00302 000627R       DEF TEMP      CARD CONTROL WORD
00303 000006X       DEF $DV19     OPTIONAL PARAMETER ADDR
00304 000353R       DEF M1        BUFFER LENGTH
*
00305 000006X READB LDA $DV19,I   GET OPTIONAL PARAMETER
00306 000366R       AND HBYTE     REMOVE HIGH BYTE
00307 002003        SZA,RSS       LOW BYTE ZERO?
00310 000344R       JMP READQ     YES, BUILD READ QUAD
00311 002021        SSA,RSS       POSITIVE NUMBER?
00312 003004        CMA,INA       YES, MAKE NEGATIVE
00313 002004        INA           SUBTRACT ONE
00314 000672R       STA IGNOR,I   SAVE IN EXTENSION
00315 000633R       JSB QUAD      BUILD READ QUAD
00316 100002X       DEF $DV15,I
00317 171600        OCT 171600    DMA CONTROL WORD
00320 100005X       DEF $DV18,I   CARD CONTROL WORD
00321 100003X       DEF $DV16,I   BUFFER ADDRESS
00322 100004X       DEF $DV17,I   BUFFER LENGTH
*
00323 000672R       LDA IGNOR,I   GET NUMBER OF INTERRUPTS TO IGNORE
00324 002003        SZA,RSS       ZERO?
00325 000335R       JMP READ1     YES, READ ONE BYTE INTO BIT BUCKET
*
```

```
00326 000633R       JSB QUAD      BUILD BIT BUCKET QUAD
00327 100002X       DEF $DV15,I
00330 071000        OCT 71000     DMA CONTROL WORD
00331 100005X       DEF $DV18,I   CARD CONTROL WORD
00332 100003X       DEF $DV16,I   BUFFER ADDRESS
00333 100672R       DEF IGNOR,I   BUFFER LENGTH
00334 000243R       JMP ID.IO     SEND DATA
*
00335 000633R READ1 JSB QUAD      BUILD READ BYTE QUAD
00336 000111R       DEF ZERO
00337 071600        OCT 71600     DMA CONTROL WORD
00340 000434R       DEF B2000     ASIC CONTROL WORD
00341 000673R       DEF BITBK     BIT BUCKET ADDRESS
00342 000353R       DEF M1
00343 000243R       JMP ID.IO     READ BYTE
*
00344 000633R READQ JSB QUAD      BUILD READ QUAD
00345 100002X       DEF $DV15,I
00346 071600        OCT 71600     DMA CONTROL WORD
00347 100005X       DEF $DV18,I   CARD CONTROL WORD
00350 100003X       DEF $DV16,I   BUFFER ADDRESS
00351 100004X       DEF $DV17,I   BUFFER LENGTH
00352 000243R       JMP ID.IO     SEND DATA
*
*
00353 177777  M1    OCT -1
00354 177776  M2    OCT -2
00355 177775  M3    OCT -3
00356 177773  M5    OCT -5
00357 000100  BIT6  OCT 100       BIT 6, "BINARY-ASCII" BIT
00360 000200  BIT7  OCT 200       BIT 7, "CRLF" BIT
00361 000400  BIT8  OCT 400       BIT 8, "CHLN" BIT
00362 001000  B1000 OCT 1000      BIT 9, "XMIT" BIT
00363 004000  BIT11 OCT 4000      BIT 11, "ESC" BIT
00364 171777  M1011 OCT 171777    MASK RCV & ECHO,  BITS (10 & 11)
00365 177400  LBYTE OCT 177400    LOWER BYTE MASK
00366 000377  HBYTE OCT 377       HIGH BYTE MASK
*
*
* CONTROL REQUEST *
*
00367 000002X CNTRL LDA $DV15,I   GET
00370 101046        LSR 6         SUBFUNCTION
00371 000406R       AND B77
00372 000403R       CPA B6        DYNAMIC STATUS?
00373 000407R       JMP DYNAM     YES
00374 000405R       CPA B43       ENABLE/DISABLE ERROR CHECKING
00375 000411R       JMP PCHK      YES
00376 000404R       CPA B23       CONTROL ASYNCHRONOUS INT.?
00377 000416R       JMP CASYN     YES
00400 002400  DONE  CLA           IGNOR REQUEST
```

```
00401 000003X       STA $DV16,I   CLEAR ERROR CODE
00402 000000R       JMP ID.00,I   INTERFACE COMPLETION
*
00403 000006  B6    OCT 6
00404 000023  B23   OCT 23
00405 000043  B43   OCT 43
00406 000077  B77   OCT 77
*
*
* DYNAMIC STATUS (FUNCTION CODE =6) *
*
00407 000532R DYNAM JSB STAT      READ ASIC STATUS & OUTPUT CNTRL WRD
00410 000400R       JMP DONE      INTERFACE COMPLETE
*
*
* ENABLE/DISABLE ERROR (FUNCTION CODE = 43) *
*
00411 000003X PCHK  LDA $DV16,I   GET PARAMETER
00412 000415R       AND PMASK     MASK PARITY & FRAMING ERROR
00413 000671R       STA PCHKB,I   SAVE IN PARITY CHECK FLAG
00414 000604R       JMP IDCOM     INTERFACE COMPLETE
*
00415 030000  PMASK OCT 30000     MASK PARITY & FRAMING ERROR BITS
*
*
* ENABLE ASYNCHRONOUS INTERRUPT (FUNCTION CODE = 23) *
*
00416 000003X CASYN LDA $DV16,I   GET PARAMETER
00417 002002        SZA           ENABLE ASYNC INT.
00420 000436R       JMP DASYN     NO
00421 000010X       LDA $IF5,I    SAVE DVT RESUME ADDR.
00422 000001X LU1   STA $IFTX,I   IN DVT EXTENSION.
*
00423 000633R EASYN JSB QUAD      BUILD READ QUAD
00424 000111R       DEF ZERO
00425 061600        OCT 61600     DMA CNTRL WRD  DONOT WRITE RESIDUE!!!
00426 000434R       DEF B2000     ASIC CONTROL WORD
00427 000673R       DEF BITBK     BIT BUCKET ADDRESS
00430 000353R       DEF M1        1 BYTE
00431 002400        CLA           DISABLE TIMEOUT
00432 000673R       STA BITBK,I   INITIALIZE BIT BUCKET
00433 000245R       JMP WDOUT     SEND DATA
*
00434 002000  B2000 OCT 2000      BIT 10, RCV
00435 000022  DC2   OCT 22        DC2 IN LOWER BYTE
*
*
* DISABLE ASYNCHRONOUS INTERRUPT (FUNCTION CODE = 23) *
*
00436 002400  DASYN CLA           ZERO DVT RESUME ADDR.
00437 000001X       STA $IFTX,I   IN DVT EXTENSION.
```

```
00440 000003X        STA $DV16,I   ZERO ERROR CODE
00441 000576R        JMP LUCHK     CHECK FOR LU=1
*
*
* CONTINUATION *
*
00442 102524  CONT   LIA 24B
00443 002003         SZA,RSS       FRONT PANEL INTERRUPT?
00444 000460R        JMP CONT1     NO, CONTINUE
00445 002400         CLA           YES, ZERO
00446 102624         OTA 24B       SELECT CODE 24
*
00447 000011X PWRFL LDB $IF6,I     GET AVAILABILITY
00450 006021         SSB,RSS       BUSY?
00451 000572R        JMP BRK       NO, CHECK FOR ASYNC CONDITION
00452 000626R        LDA BREAK     DON'T DOWN/DON'T FLUSH, RESTART NO MESS
00453 000003X        STA $DV16,I   ERROR CODE
00454 107721         CLC 21B,C     SUSPEND AND
00455 107723         CLC 23B,C     TERMINATE DMA OPERATION
00456 000532R        JSB STAT      READ ASIC STATUS & OUTPUT CNTRL WRD
00457 000604R        JMP IDCOM     INTERFACE COMPLETE
*
00460 102222  CONT1  SFC 22B       DMA COMPLETION?
00461 000014X        JMP $DMPR     NO, MEMORY ERROR
00462 000011X        LDA $IF6,I    GET AVAILABILITY
00463 002020         SSA           BUSY?
00464 000541R        JMP TICST     YES
00465 000000R        ISZ ID.00     NO, SETUP FOR CONTINUE
00466 000001X        LDA $IFTX,I   GET DVT RESUME ADDR.
00467 002002         SZA           ASYNCHRONOUS INT. ENABLED?
00470 000476R        JMP CONT4     YES
00471 107730         CLC 30B,C     CLEAR INTERRUPT FLAG
00472 107721         CLC 21B,C     SUSPEND AND
00473 107723         CLC 23B,C     TERMINATE DMA OPERATION
00474 000607R        LDA B4        REPORT AN ILLEGAL INTERRUPT
00475 000000R        JMP ID.00,I   INTERFACE CONTINUE
*
00476 000010X CONT4  STA $IF5,I    SAVE DVT RESUME ADDR.
00477 000673R        LDB BITBK,I   CHECK IF BLOCK MODE ENABLED
00500 101050         LSR 8         SHIFT TO LOWER BYTE
00501 000435R        CPB DC2       BLOCK MODE?
00502 000505R        JMP HOLD      YES
00503 000000R        ISZ ID.00     DEVICE RESUME
00504 000423R        JMP EASYN     ENABLE ASYNCHRONOUS INT.
*
00505 000633R HOLD   JSB QUAD      BUILD DC1 QUAD
00506 000111R        DEF ZERO
00507 171400         OCT 171400    DMA CONTROL WORD
00510 000362R        DEF B1000     ASIC CONTROL WORD
00511 000713R        DEF DC1A      DC1 ADDRESS
00512 000353R        DEF M1        1 BYTE
```

```
00513 000633R       JSB QUAD      BUILD READ QUAD
00514 000111R       DEF ZERO
00515 171600        OCT 171600    DMA CONTROL WORD
00516 000434R       DEF B2000     ASIC CONTROL WORD
00517 000673R       DEF BITBK     BIT BUCKET ADDRESS
00520 000353R       DEF M1        1 BYTE
00521 000633R       JSB QUAD      BUILD 'DC1' QUAD
00522 000111R       DEF ZERO
00523 071400        OCT 71400     DMA CONTROL WORD
00524 000362R       DEF B1000     ASIC CONTROL WORD
00525 000713R       DEF DC1A      BUFFER ADDRESS
00526 000353R       DEF M1        BUFFER LENGTH
*
00527 000606R       LDA B3        ASSERT HOLD & TIMEOUT
00530 000673R       STA BITBK,I
00531 000245R       JMP WDOUT     SEND DATA
*
*
* READ ASIC STATUS *
*
00532 000000  STAT  NOP           READ ASIC STATUS & OUTPUT CNTRL WRD
00533 102531        LIA 31B       READ OUTPUT CONTROL WORD
00534 000006X       STA $DV19,I   SAVE IT
00535 102532        LIA 32B       READ ASIC STATUS WORD
00536 000670R       IOR BRKFL,I   MERGE BREAK FLAG INTO STATUS
00537 000005X       STA $DV18,I   SAVE IT
00540 000532R       JMP STAT,I    RETURN
*
00541 000532R TICST JSB STAT      READ ASIC STATUS & OUTPUT CNTRL WRD
00542 002020        SSA           VAL DATA BIT SET?
00543 000554R       JMP TLOG      YES, IGNOR ERROR BITS
00544 001200        RAL
00545 002020        SSA           BREAK BIT SET?
00546 000260R       JMP WAIT      YES, WAIT FOR DMA COMPLETION
00547 000625R       AND EMASK     CHECK FRAMING, PARITY & OVERRUN
00550 002003        SZA,RSS       ZERO?
00551 000554R       JMP TLOG      YES, NO ERROR
00552 000610R       LDB B5        NO, TRANSMISSION ERROR
00553 000616R       JMP TDMA      TERMINATE DMA OPERATION
*
00554 103123  TLOG  CLF 23B       CLEAR FLAGS 20, 21 & 22
00555 102523        LIA 23B       READ REMAINING CHARACTERS (NEG)
00556 000004X       LDB $DV17,I   GET BUFFER LENGTH
00557 006020        SSB           ARE THEY CHARACTERS?
00560 007005        CMB,INB,RSS   YES
00561 005000        BLS           MULTIPLY WORDS BY 2
00562 000001        ADA B         FIND ACTUAL CHARACTER COUNT (POS)
00563 000004X       LDB $DV17,I   GET BUFFER LENGTH
00564 006021        SSB,RSS       ARE THEY CHARACTERS?
00565 001100        ARS           NO, DIVIDE CHARS. BY 2
00566 006400        CLB
```

```
00567 000004X       STA $DV17,I   SAVE AS + CHARS OR + WORDS
00570 000003X       STB $DV16,I   SETUP ERROR CODE
00571 000573R       JMP ASYNC     ENABLE ASYNCHRONOUS INTERRUPT
*
00572 000000R BRK   ISZ ID.00     EXIT WAIT FOR VALID REQUEST
*
00573 000001X ASYNC LDB $IFTX,I   GET DVT RESUME ADDR
00574 006002        SZB           WAS ASYNCHRONOUS INT. ENABLED?
00575 000423R       JMP EASYN     YES, RE-ENABLE INTERRUPT
00576 000010X LUCHK LDA $IF5,I
00577 000013X       CPA $LUTA,I   LU=1?
00600 000422R       JMP LU1       YES, RE-ENABLE INTERRUPT
00601 002400        CLA
00602 102631        OTA 31B       DISABLE PE & OE INT'S
00603 107730        CLC 30B,C     DISABLE BREAK
00604 002400  IDCOM CLA
00605 000000R       JMP ID.00,I   INTERFACE COMPLETION
*
00606 000003  B3    OCT 3
00607 000004  B4    OCT 4
00610 000005  B5    OCT 5
*
*
* ZERO TRANSMISSION LOG *
*
00611 000003X ZLOG  STB $DV16,I   ERROR CODE
00612 000532R       JSB STAT      READ ASIC STATUS & OUTPUT CNTRL WRD
00613 000573R       JMP ASYNC     ENABLE ASYNCHRONOUS INTERRUPT
*
*
* TIME-OUT *
*
00614 000606R TIMOT LDB B3        TIME-OUT ERROR.
00615 000532R ABORT JSB STAT      READ ASIC STATUS & OUTPUT CNTRL WRD
00616 000003X TDMA  STB $DV16,I   CREATE ERROR CODE
00617 006400        CLB
00620 000004X       STB $DV17,I   ZERO TRANSMISSION LOG
00621 107721        CLC 21B,C     SUSPEND AND
00622 107723        CLC 23B,C     TERMINATE DMA OPERATION
00623 106632        OTB 32B       RESET ASIC CARD
*
* NOTE: FOR , FUTURE REFRENCE, >100MS. MUST BE ALLOWED BETWEEN
*       A CARD RESET AND OUTPUT DMA (RESET DOESN'T COMPLETE FOR
*       18 CYCLES)
*
00624 000573R       JMP ASYNC     ENABLE ASYNCHRONOUS INTERRUPT
*
*
00625 070000  EMASK OCT 70000     MASK FRAMING, PARITY & OVERRUN
00626 100077  BREAK OCT 100077    DON'T DOWN/DON'T FLUSH, RESTART NO MESS
00627 000000  TEMP  NOP           TEMPORARY STORAGE
```

```
00630 006412  CRLFX OCT 6412       'CRLF'
00631 015537  ESCX  OCT 15537      'ESC'
00632 010400  DC1X  OCT 10400      'DC1'
*
*
* BUILD DATA QUAD *
*
00633 000000  QUAD  NOP
00634 000633R       LDA QUAD,I
00635 000007X       LDB $IF1
00636 000633R       ISZ QUAD
00637 000015X       JSB $SELR      SET RELOCATION REGISTER
00640 000633R       IOR QUAD,I     MERGE RELOCATION REG. NUMBER
00641 000667R       ISZ DMAAD
00642 000667R       STA DMAAD,I    DMA CONTROL WORD
00643 000655R       JSB NEXT       ASIC CONTROL WORD
00644 000655R       JSB NEXT       BUFFER ADDRESS
00645 000655R       JSB NEXT       BUFFER LENGTH
00646 000633R       ISZ QUAD       FIX RETURN ADDRESS
00647 006020        SSB            CHARACTERS?
00650 000633R       JMP QUAD,I     YES, QUAD COMPLETE
00651 005000        BLS            NO, SAVE
00652 007004        CMB,INB        BUFFER LENGTH
00653 000667R       STB DMAAD,I    IN CHARACTERS
00654 000633R       JMP QUAD,I     QUAD COMPLETE
*
*
00655 000000  NEXT  NOP
00656 000667R       ISZ DMAAD
00657 000633R       ISZ QUAD
00660 000633R       LDB QUAD
00661 000001        LDB B,I
00662 005275        RBL,CLE,SLB,ERB
00663 000661R       JMP *-2
00664 000001        LDB B,I
00665 000667R       STB DMAAD,I
00666 000655R       JMP NEXT,I
*
*
00667 000000  DMAAD NOP            DVT RESUME ADDR PTR
00670 000000  BRKFL NOP            BREAK FLAG
00671 000000  PCHKB NOP            PARITY CHECK FLAG
00672 000000  IGNOR NOP            IGNORE INPUT FLAG
00673 000000  BITBK NOP            BIT BUCKET
*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
00674 000000  WD18A NOP            MODEM STATUS WORD ADDRESSES       *BC*
00675 000000  WD19A NOP            RETRY RESUME ADDRESS              *BC*
00676 000000  WD20A NOP            \                                 *BC*
00677 000000  WD21A NOP             > MODEM ALARM PROGRAM NAME       *BC*
00700 000000  WD22A NOP            /        ADDRESSES                *BC*
00701 000000  WD23A NOP            ALARM PROG LOGLU  ADDRESS         *BC*
```

```
00702 000000  SAVEA NOP                                           *BC*
00703 000000  STSSS NOP           STATUS SNAP SHOT ADDRESS        *BC*
00704 000000  RQ    NOP           REQUEST "    "      "           *BC*
00705 000000  W18   NOP           WD18    "    "      "           *BC*
00706 000000  DIREC NOP           ENTRY DIRECTIVE     "           *BC*
00707 000000  STRA  NOP           TEMP STORAGE        "           *BC*
00710 000000  STRB  NOP             "     "           "           *BC*
00711 000000  CRLFA NOP             "     "                    A.83BC
00712 000000  ESCA  NOP             "     "                    A.83BC
00713 000000  DC1A  NOP             "     "                    A.83BC
*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*_*
*
* INTERFACE STORAGE *
*
*     WORD  1:  DVT RESUME ADDRESS
*     WORD  2:  DMA CONTROL WORD       \
*     WORD  3:  CARD CONTROL WORD       \ 1ST QUAD
*     WORD  4:  BUFFER ADDRESS          /
*     WORD  5:  - BUFFER LENGTH (CHAR) /
*     WORD  6:  DMA CW                 \
*     WORD  7:  CARD CW                 \ 2ND QUAD
*     WORD  8:  BUF ADDR                /
*     WORD  9:  BUF LENGTH             /
*     WORD 10:  DMA CW                 \
*     WORD 11:  CARD CW                 \ 3RD QUAD
*     WORD 12:  BUF ADDR                /
*     WORD 13:  BUF LENGTH             /
*     WORD 14:  BREAKFLAG
*     WORD 15:  ERROR CHECKING FLAG
*     WORD 16:  IGNORE INPUT FLAG
*     WORD 17:  BIT BUCKET
*     WORD 18 -33 FOR ID.01 COMPATABILITY
*
* DRIVER PARAMETER STORAGE *
*
*     NONE
*
*
00714 174377
            END
```

```
Macro/1000 Cross reference

* - Volatile reference (store, jump, call...)

$DIOC  . . . . . . . . 19:    95*
$DMPR  . . . . . . . . 19:   416*
$DV15  . . . . . . . . 18:   120    129    145    153    161    168    175
                             182    201    207    280    291    307    331
$DV16  . . . . . . . . 18:   164    185    204    283    294    310    341*
                             358    368    392*   409*   496*   520*   529*
$DV17  . . . . . . . . 18:   146    165    186    205    284    311    486
                             491    495*   531*
$DV18  . . . . . . . . 18:   127    144*   163    184    191    203    215
                             223    260    282    293    309    469*
$DV19  . . . . . . . . 18:   256    268    271    466*
$DVTP  . . . . . . . . 19:   136
$IF1 . . . . . . . . . 19:   555
$IF5 . . . . . . . . . 19:    94    371    430*   504
$IF6 . . . . . . . . . 19:   405    417
$IFTX  . . . . . . . . 18:    28    239    372*   391*   421    501
$LUTA  . . . . . . . . 19:   505
$SELR  . . . . . . . . 19:   557*
A  . . . . . . . . . . 24:   Symbol not referenced
ABORT  . . . . . . . .528:    88*
ABRTE  . . . . . . . .106:    86
ASBLK  . . . . . . . .198:   171*
ASCII  . . . . . . . .168:   156*
ASYNC  . . . . . . . .501:   497*   522*   540*
B  . . . . . . . . . . 25:    72*    77*    82*   490    577    580
B1 . . . . . . . . . .107:    90
B1000  . . . . . . . .322:   130    262    441    453
B2 . . . . . . . . . .108:    97
B2000  . . . . . . . .384:   301    377    447
B23  . . . . . . . . .345:   338
B3 . . . . . . . . . .513:    99    121    122    457    527
B3000  . . . . . . . .110:   132
B4 . . . . . . . . . .514:   427
B43  . . . . . . . . .346:   336
B5 . . . . . . . . . .515:   481
B6 . . . . . . . . . .344:   334
B7 . . . . . . . . . .109:    85
B77  . . . . . . . . .347:   333
BINRY  . . . . . . . .158:   178*
BIT11  . . . . . . . .323:   169
BIT6 . . . . . . . . .319:   154
BIT7 . . . . . . . . .320:   176    208
BIT8 . . . . . . . . .321:   142
BITBK  . . . . . . . .589:    37*   302    378    381*   431    448    458*
BREAK  . . . . . . . .544:   408
BRK  . . . . . . . . .499:   407*
BRKFL  . . . . . . . .586:    31*   247*   468
```

```
CASYN  . . . . . . . .368:  339*
CNTRL  . . . . . . . .331:  123*
CONT . . . . . . . . .399:   98*
CONT1  . . . . . . . .415:  401*
CONT4  . . . . . . . .430:  423*
CRLF1  . . . . . . . .188:  180*
CRLFA  . . . . . . . .604:   70*    192    216
CRLFQ  . . . . . . . .207:  199*
CRLFX  . . . . . . . .546:   71
D13  . . . . . . . . .111:   30
DASYN  . . . . . . . .390:  370*
DC1A . . . . . . . . .606:   80*    442    454
DC1X . . . . . . . . .548:   81
DC2  . . . . . . . . .385:  433
DIREC  . . . . . . . .601:   62*     92*    96
DMAAD  . . . . . . . .585:   29*    559*   560*   569*   574*   581*
DONE . . . . . . . . .340:  353*
DYNAM  . . . . . . . .352:  335*
EASYN  . . . . . . . .374:  436*    503*
EMASK  . . . . . . . .543:  478
ESCA . . . . . . . . .605:   75*    224
ESCX . . . . . . . . .547:   76
HBYTE  . . . . . . . .326:  272
HOLD . . . . . . . . .438:  434*
ID.00  . . . . . . . . 27:  231*    243*   248*   249*   342*   420*   428*
                            435*    499*   511*
ID.IO  . . . . . . . .230:  166*    194*   218*   296*   304*   312*
IDCOM  . . . . . . . .510:  361*    413*
IGNOR  . . . . . . . .588:   35*    119*   278*   286    295
INIT . . . . . . . . .118:   91*
LBYTE  . . . . . . . .325:  257
LU1  . . . . . . . . .372:  506*
LUCHK  . . . . . . . .504:  393*
M1 . . . . . . . . . .315:  269     303    379    443    449    455
M1011  . . . . . . . .324:  261
M2 . . . . . . . . . .316:  193
M3 . . . . . . . . . .317:  225
M5 . . . . . . . . . .318:  217
NEXT . . . . . . . . .573:  561*    562*   563*   582*
NOCR . . . . . . . . .220:  210*
PCHK . . . . . . . . .358:  337*
PCHKB  . . . . . . . .587:   33*    140    360*
PMASK  . . . . . . . .363:  359
PWRFL  . . . . . . . .405:  104*
QUAD . . . . . . . . .553:  160*    181*   188*   200*   212*   220*   264*
                            279*    290*   298*   306*   374*   438*   444*
                            450*    554    556*   558    564*   566*   570*
                            575*    576
READ . . . . . . . . .254:  148*
READ1  . . . . . . . .298:  288*
READB  . . . . . . . .271:  259*
```

```
READQ . . . . . . . .306:   274*
RQ . . . . . . . . . .599:    58*
SAVEA . . . . . . . .597:    54*
SET7 . . . . . . . . .144:   141*
SET8 . . . . . . . . .142:   135*    139*
STAT . . . . . . . . .464:   352*    412*    470*    472*    521*    528*
STRA . . . . . . . . .602:    64*
STRB . . . . . . . . .603:    66*
STSSS . . . . . . . .598:    56*
T  . . . . . . . . . .113:    68*     84
TDMA . . . . . . . . .529:   482*
TEMP . . . . . . . . .545:   263*    267
TICST . . . . . . . .472:   419*
TIMOT . . . . . . . .527:   100*
TLOG . . . . . . . . .484:   474*    480*
W18 . . . . . . . . .600:    60*
WAIT . . . . . . . . .245:   477*
WD18A . . . . . . . .591:    42*
WD19A . . . . . . . .592:    44*
WD20A . . . . . . . .593:    46*
WD21A . . . . . . . .594:    48*
WD22A . . . . . . . .595:    50*
WD23A . . . . . . . .596:    52*
WDOUT . . . . . . . .232:   382*    459*
ZERO . . . . . . . . .112:   189     213     221     265     299     375     439
                            445     451
ZLOG . . . . . . . . .520:   159*    255*
Macro:  No  errors total
```

Chapter 5
General Driver Concerns

I/O Request Parameters

The I/O request parameters issued by the user are supplied to the driver in the DVT as shown below. The driver parameter area of the DVT may also contain information about the device that is not specific to the current request. If the interface driver is being called, the driver communication flags in DVT20 may also have meaning.

```
       +------------------------------------------------+
DVT15  |        | Z|   Subfunction    |      |L |  | RQ |
       |-----+--+--+------------------+------+--+--+----|
DVT16  |              Request Parameter 1               |
       |------------------------------------------------|
DVT17  |              Request Parameter 2               |
       |------------------------------------------------|
DVT18  |              Request Parameter 3               |
       |------------------------------------------------|
DVT19  |              Request Parameter 4               |
       +------------------------------------------------+
```

DVT15 is the control word for an I/O request. The Z bit interacts with the RQ bits, and is described below with RQ.

The SUBFUNCTION format in DVT15 is:

```
+-----------------------------+
| 11   10    9    8    7    6 |
|----+----+----+----+----+----|
| X  | TR | X  | EC | X  | BI |
+-----------------------------+
```

The bits marked "X" are driver-defined.

TR is transparency mode. 0 is off; 1 is on.

EC is to echo input: 0 indicates no echo; 1 sets echo on.

BI is the data format: 0 indicates ASCII; 1 indicates binary.

Bits 11 through 6 (all SUBFUNCTION bits) must be set to 1 if and only if the device type is 30-37 (discs).

BI and TR operate together to specify a set of data handling circumstances for special characters and EOR (end of record) processing. These conventions are explained in the Driver Reference Manual.

It is not necessary for all drivers to support the full set of variances possible. However, when it is desirable to handle one or more of these conditions, they should be implemented according the beyond those described can be controlled by the X bits.

L is the mapping location of the buffer. 0 indicates the system map; 1 indicates the user map. L=1 may also indicate the System Available Memory (SAM) map. Drivers must, therefore, never try to find data buffers on their own. They should use $READ/$WRIT or $ONER/$ONEW. See Chapter 7 for more information about this pair of routines.

In DVT15, RQ is the request code itself. It equals 1 for a read request, 2 for write, and 3 for control.

The Z bit, when set, indicates that Parameters 3/4 describes a buffer/buffer length. The Z bit may be used for any RQ (1, 2 or 3).

The interaction between the Z bit and the request code RQ is:

	RQ = 1 or 2		RQ = 3	
	Z = 0	Z = 1	Z = 0	Z = 1
Parm 1	Buf Addr	Buf Addr	Simple Var	Simple Var
Parm 2	Buf Len	Buf Len	Simple Var	Simple Var
Parm 3	Simple Var	Buf Addr	Simple Var	Buf Addr
Parm 4	Simple Var	Buf Len	Simple Var	Buf Len

For an RQ of 1 or 2 (read or write), Parameters 1 and 2 describe an input buffer in which the driver transfers data. Parameter 1 is the data buffer address and Parameter 2 is the length of the buffer. If data is to be accessed in the buffer, the $READ and $WRIT subroutines must be used.

For an RQ of 3 (control) or 0 (multibuffered request), the user-specified buffer provides information to be acted upon by the driver. (An RQ of 0 is covered in the Device/Interface Driver Interactions chapter, under the Multibuffered Request section.)

To demonstrate some of the possible usages of these optional parameters:

The DD.00 terminal device driver supports a WRITE/READ request. If RQ is 1 (read) and Z is 1:

Parameter 1 is the input buffer address
Parameter 2 is the input buffer length
Parameter 3 is the output buffer address
Parameter 4 is the output buffer length

DD.30 is the disc device driver. It uses optional parameters to define track and sector. If RQ is 1 or 2 (read or write) and Z is 0:

Parameter 1 is the input buffer address
Parameter 2 is the input buffer length
Parameter 3 is the track
Parameter 4 is the sector

Zero-Length Requests

As a general rule, a zero length request should provide the end-of-record handling condition, which would normally be supplied if data were actually transfered. Thus, according to the TR/BI modes of operation, the general circumstances are as follows. As before, drivers are expected to support these operations only where useful.

TR	BI	Action on Input	Action on Output
0	0	Return zero transmission log and exit.	Issue CRLF and/or EOR line signal.
0	1	Same as above.	Issue EOR line signal, if available.
1	0	Same as above.	Return zero transmission log and exit.
1	1	No operation.	No operation.

Illegal Requests

Illegal requests are generally handled according to the following rules:

1. If a driver receives an illegal READ/WRITE requests, the standard procedure to reject the request is:

   a. Set error code 1 in DVT16.
   b. Make a "done" exit, which completes the request.

2. If a driver receives an unsupported zero length read/write request, the driver should ignore the request.

   a. Set error code 0 in DVT16.
   b. Make a "done" exit, which completes the request.

3. Unsupported control requests should be handled in the same way as unsupported zero length read/write requests.

Posting Status

Status can have several meanings:

1. Status associated with the request.

   On a read, as an example, a program may need to know how many bytes or words were read. This number is the transmission log. The transmission log is posted in DVT17, generally by the interface driver.

2. Status associated with the device.

   For example, a cassette tape may be at the end of the useable area after a request.

3. There is a possible error associated with the request, even if it completes successfully. For example, the request may have succeeded after a number of attempts (as determined by the driver).

Errors will be covered partially in this section and in more detail in the following section.

Status is posted in the DVT upon completion of a request. Either the device driver or the interface driver may post status; any non device-dependent status is posted by the interface driver. It is also important to remember that an interface driver may be called directly by a program and so it should post as much information as possible.

There are two places in the DVT to post status: bits 1-7 of DVT6 (the status byte) and all of DVT18 and DVT19 (the extended status). The bits in DVT6 are general in nature and may be interpreted generally without regard to the actual device. DVT18 and DVT19, however, provide device-dependent information or else may help interpret or modify the meaning of the bits in DVT6. The status byte in DVT6 includes an error bit (E, bit 0) which is controlled by the system, not the driver. The E bit is set if the driver posts an error code in DVT16 prior to exit. The E bit is cleared on initiation of a new request.

The extended status words may provide detail about an error condition. For example, the error code in DVT16 may be a 7 (address error), while information in DVT18 might indicate that the cause of the address error was an incorrect sector address. The extended status could also be used to record operable but degrading device conditions, such as seek retry counts, etc.

The format of the DVT6 status byte is given below. There is no defined format for the extended status words DVT18 and DVT19.

7	6	5	4	3	2	1	0	Bit number
EOF	DB	EOM	BOM	SE	DF	DF	E	Mnemonic

Bits 7 through 0 are set by the driver as needed.
The E bit is set by the system.

EOF is End Of File. Use for mini-cassette tapes, card readers, etc. EOF = 1 when condition is true.

DB is Device Busy. Indicates that the device is performing a function which prevents other operations from starting, such as tape rewind. DB = 1 when condition is true.

EOM is End Of Medium. Set when the current request has positioned (or will position) the physical medium past the maximum limit (for instance, trying to write 2 disc tracks when only 1 track remains for use).

> NOTE: If the EOM bit is set, it is generally a good idea to set the EOF bit also. This ensures FMP compatibility.

BOM is Beginning of Medium. When set, indicates that the medium is at the start of the recording area.

SE is Soft Error. An error occurred which caused the driver to attempt an error recovery operation. The E bit may or may not be set, depending upon whether or not the operation was eventually successful.

DF is Driver Definable.

E is an Error Indicator set by system if the driver sets any error code in DVT16. Drivers should not change this bit.

The status byte is accessible to a program by either making an exec request (an EXEC 3 on LU+600B) or by checking the A-register after non-buffered requests.

Extended status is recovered through a call to RMPAR immediately following a non-buffered request.

Posting Errors

Errors are reported in the DVT by the device driver, or the interface driver, or both, according to the design on the drivers. If you are writing a device driver to work with an existing interface driver (or vice versa), you must have a knowledge of how the other driver interacts with the DVT.

Drivers report errors by storing error codes in DVT16 bits 0-5. After the driver exits, the system will check these bits and, if they are not zero, will set the E bit (any error) in DVT6.

Error codes 1-12 will result in pre-defined error mnemonics being issued by the system. Otherwise, the system will merely report the error number, which may be unique to the device. The error code is also accessible programmatically with a status request.

The default action taken by the system is to down the DVT on the error exit, making it unavailable for new requests until it is upped. However, the driver can override the default by setting bit 15 in DVT16, as illustrated below.

When a down DVT is brought up, the request which caused the error is normally re-initiated on the device driver. The driver may also override this restart by setting bit 14 in DVT16. This will cause the request to be flushed from the I/O queue (removed from the linked request list).

DVT16 format upon driver exit:

```
+---------------------------------+
¦15   14            5  4 ... 1   0¦
¦---+---+-----------+-------------¦
¦ D ¦ F ¦           ¦ Error Code  ¦
+---------------------------------+
```

D is the DVT down bit. If D=1, then the DVT is not set down on an error; if D=0, the DVT is set down on an error.

F is the flush bit. 1 indicates flush the request; 0 means don't flush it.

Default: Both bits zero (set the DVT down and do not flush the request.

The D and F bits will be ignored by the system if the error code is zero.

Other combinations of the D and F bits and their meanings are:

D F

0 1 Set the device down, and flush the request. The request is "finished" in the sense that the system will not repeat the request on the driver when the device is upped. This implies that the request actually successfully completed. However, during its activity some other error circumstance was discovered which requires operator intervention.

1 1 Do not set the device down but flush the request. This is provided as a soft error reporting condition. The request has completed successfully after having some difficulty (such as disc retries). This condition is also forced by the system if the caller set the UE bit (normal requests only).

1 0 Do not set the device down, and do not flush the request. The resulting action is to restart the request. This, in combination with error code 63 (77 octal) is used in driver-directed power fail recovery. The request is automatically restarted by a new initiate entry.

The case where both bits are zero is most common. For example, if a line printer is out of paper, it is best to set the device down but not flush the request. When the device is set down, the program making the request is suspended.

When new paper is installed, the operator can set the device up and the request will complete. Programs which make new requests on the device while it is down will be suspended until the condition is corrected and the device is upped.

The device driver may put itself in the time list and up itself when it finds the condition corrected. See section on system callable routines.

A driver would normally instruct the system to flush the request only if the request was illegal (such as illegal track specified on a disc read/write), since re-initiating the request would lead to the same error.

The effect of setting the error code in bits 5-0 of DV16 is shown in Table 5-1 below.

Table 5-1. Error Codes and their Meanings

Error Code (Decimal)	Meaning
0	No Error
1	Illegal Request
2	Not Ready
3	Time Out
4	End of Tape
5	Transmission Error
6	Write Protected
7	Address Error
8	Serial Poll Failure (HPIB)
9	Group Poll Failure (HPIB)
10	Fault
11	Data Communication Error
12	Generation Error
13 to 20	Reserved
21 to 59	Driver Definable
60 to 62	Reserved
63	Restart if D=1, F=0

Error code 63 is unique, since it permits the request to be re-initiated with no error message even though the driver made an error exit. However, to cause automatic restart, the driver must set the D bit (do not set the DVT down).

The following message is reported when a device error occurs. The first two lines always appear. The last two appear only when the device is set down or the request is flushed.

```
I/O Device Error on LUnn   The reason is:

<meaning>  (From Table 5-1)
Device has been downed (use UP to try recovery)
Request has been flushed
```

Driver Partitioning

In order for a driver to be generated into a partition during system generatio n, it must contain the gen record GEN PARTITIONABLE. This record implies that the driver will be mappable during system execution. The term mappable means that the driver does not perform DMA to or from its code space. It is, however, proper to perform DMA from a table, such as the DVT or IFT, or from the user buffer. The DMA control words must not be part of the driver's code space. They should be in the DVT or IFT extension area,

Chapter 6
Device and Interface Driver Interactions

Parameter Passing Between Drivers

All communication between device/interface driver pairs is via the DVT. Information passed to the device driver about the user's request is contained in DVT15-DVT19. The device driver will examine the request parameters and may replace one or more of them with its request on the interface driver.

When the interface driver completes, it posts status in DVT6, error code (if any) in DVT16 and transmission log (positive number of bytes or words transmitted) in DVT17. Extended status information (if any) should be stored in DVT18 and DVT19. The device driver may again modify the DVT before completing the request. For example, the extended status information may have more meaning when considered in terms of the actual device and may cause the device driver to reinitiate the request on the interface driver. This permits the device driver to handle error recovery procedures.

Generally, the interface driver should operate on the DVT as if the device driver did not exist. (This may not be feasible in some cases.) If there is a need for direct communication between the device driver and the interface driver, this is accomplished through the driver communication bits in DVT20.

Multibuffered Request

The multibuffered request (RQ=0) is a chain of requests built by the device driver and passed to the interface driver. The device driver will never receive a request of this type. This capability permits the device driver to break up a complex request into a series of simpler operations.

The interface driver is initiated once, to begin the chain and the request is not complete until the chain completes. The interface driver uses the chain to build a chained DMA request. For a discussion of DMA chaining, see the chapter on I/O Card Processing.

The format of the multi-buffered request is defined totally by the interface driver and no standard is enforced regarding format. In the following discussion, an approach is discussed but other approaches are equally valid.

The device driver builds the request chain in the DVT extension and sets the RQ field (DVT15) to 0 prior to initiating the interface driver. The address of the chain (in this case the DVT extension) is put into DVT16 and the total length of the chain in DVT17 as a negative link count. DVT18 and DVT19 may be used to pass additional control information to the interface driver.

When a device driver is making multibuffered request links, it is responsible for setting the L bit to the correct value for each link. Each link normally corresponds to 1 DMA request. If a link represents the request data buffer, then the L bit should be set to the same value as the L bit in DVT15. If a link represents data from the driver's own area (the driver extension area) then the L bit should be set to zero.

Each link in the chain uses the following format:

```
 ---------------------------------------------------
¦15¦14¦13¦12¦11¦10¦ 9¦ 8¦ 7¦ 6¦ 5¦ 4¦ 3¦ 2¦ 1¦ 0¦       |
¦--------+--+----------------+-----+--+--+-----¦       |
¦ LEN    ¦ Z¦  Subfunction   ¦ 0  0¦ L¦ 0¦ RQ  ¦       |
¦--------+--+----------------+-----+--+--+-----¦       |
¦              Buffer Address                  ¦       |
¦----------------------------------------------¦       } LINK
¦              Buffer Length                   ¦       |
¦----------------------------------------------¦       |
¦              Control Word                    ¦       |
¦----------------------------------------------¦       |
¦              Control Word                    ¦       |
 ---------------------------------------------------
```

LEN is the length of link in chain (1 to 5 words).

RQ is the request type for this link (1 for read, 2 for write, and 3 for control).

Z is the control buffer bit. If present, 4th and 5th word of link are address and length of a control buffer.

L is the location (system/user) bit, defining whether data is in the system or user map.

I/O Table Reference

The system sets up pointers to the words in the DVT prior to entering the device driver. Also it sets up pointers to the IFT prior to entering the interface driver.

There are times however when the device driver will wish to reference entries in the IFT or the interface driver will wish to reference entries in the DVT. To reduce overhead, the system does not set up the pointers for such a cross reference (except on Interface Initiate and Abort) and so the driver must do this for itself.

A system subroutine, $DIOC, facilitiates access by a driver to any DVT or IFT. This routine can be called by either the device or the interface driver. After calling this routine with the appropriate control parameters, the driver may directly access the DVT or IFT words by loading the word indirectly. See the chapter on Callable System Routines for more information on $DIOC.

If a single address is all that is needed, it may be more efficient for the driver to compute the address needed rather than call upon $DIOC.

Asynchronous I/O and Polling

In any truly asynchronous transfer, the time interval between operations is variable and may be quite lengthy.

One such instance is a read or write request to a disc. The request will have two basic components: the seek-to-cylinder and the actual transfer of data. If the objective is to use the I/O card efficiently in the case where there may be many devices on the bus (as an HP-IB), then it is desirable to permit additional requests to be handled by the interface driver between the seek and the data transfer.

Another instance is in the polling of devices on the bus. It is desirable to initiate a poll request to several devices as quickly as possible, without waiting for each device to respond. Then the responses can be serviced as they come in. In the meantime, other requests should be permitted on the bus.

A third example applies to handling of terminals. A terminal should be able to respond an operator attention key and also be used for programmatic I/O. However, a terminal must be specifically enabled to permit recognition of an operator attention key. The enable request must complete so that programmatic requests are not held off.

To implement this feature, the interface driver must accomplish a "pseudo done" exit, which must be recognized by a co-operating device driver. The device driver then waits for "true done" (the seek has completed or the polled device responded).

For "pseudo done" the interface driver adds to an internal list the DVT address which must be used when the device makes a response. It then makes a "done" exit. As far as the system is concerned, the request is actually done and new requests may be started on the interface driver - this is important to keep the interface card as busy as possible.

In making the "done" exit, the interface driver may set the D bit in the system flags to defer calling the device driver. This would introduce the necessity for the interface driver to manage the time-outs, however. The default condition (D bit clear) will result in the continue entry to the device driver whose DVT is given in IFT5 and is a simpler situation for the driver to handle.

When an interrupt occurs and the device driver is entered, it recognizes, by the nature of the request and (possibly) the driver communication flags in DVT20, that the request is either actually complete or still in progress. If not complete, then the device driver makes a "wait" exit.

When the device makes the desired response, the interface driver consults its list to identify the device driver that should handle the request. It then places the correct DVT address in IFT5. It may then call system subroutine $DIOC to set up the pointers to the DVT and, possibly, set a flag in the driver communication area. It, again, makes a "done" exit, which causes a continue entry to the same device driver as previously. This time, however, the device driver handles it as a completion of the original request and makes a device done exit. (See the Callable System Routines for more on $DIOC.)

The device done exit causes the rescheduling of any program waiting upon the request. Therefore, there is no "pseudo done" for the device driver.

Chapter 7
Callable System Routines

$DIOC: Set Up DVT or IFT

Subroutine $DIOC may be used by either a device driver or an interface driver as follows:

```
+---------------------------+
¦ A Reg bits = ¦ 2 ¦ 1 ¦ 0 ¦    A = Advance DVT ref at IFT5
¦--------------+---+---+---¦    I = Set up IFT address pointers
¦              ¦ A ¦ I ¦ D ¦    D = Set up DVT address pointers
+---------------------------+
```

```
B Reg = DVT Address

JSB $DIOC
Return: P+1    Registers meaningless
```

If used by the device driver then only the I bit makes sense on entry to $DIOC. The device driver might wish to set up the IFT pointers so that it could place some value(s) in the IFT extension (for example) prior to an "initiate" exit.

If used by the interface driver, then either "A" or "D" may be used or both may be set. "I" may be set but makes no sense (because the IFT addresses are already set up). If both "A" and "D" are set, the advance to the next DVT in the circular list is made prior to setting up the DVT pointers, so that the pointers refer to the next DVT. "A" by itself merely changes the contents of IFT5 to the address of the next DVT.

Since the circular list pointer will point to itself if there is no circular list, the routine will work properly even if there is only one DVT attached to the IFT.

$DVLU: Compute LU From DVT

$DVLU finds the first logical unit number associated with the DVT. The calling sequence is:

```
    B-register = DVT Address

    JSB $DVLU
    Return: P+1

    A-register has LU number (or zero if no LU assigned)
    B and E registers unchanged
```

$UPIO: Up Device

$UPIO is not a closed subroutine; it is accessed by a JMP instruction rather than a JSB.

The driver jumps to $UPIO to "up" a device whose DVT pointers have been previously set up. All programs waiting on the downed device will be rescheduled by the system.

All requests in the queue will be allowed to continue.

NOTE

> Drivers should not jump to $UPIO if the device is busy. Also, they should ensure that "hold" in the system flags was not left set from a previous exit. This is to ensure that the driver will be re-entered.

$DMPR: DMA Parity Error

If a DMA parity error is received by the driver, it may enter the system with a JMP to $DMPR to allow the system to process the error.

If the parity error occurs in the operating system area, the system will execute a HLT instruction. The A-register will contain the failing page address, and the B-register will contain the physical page number.

If the parity error occurs in a user partition and the error is a hard parity error, the partition is downed, a message is given to the system console, and processing resumes.

$XQSB: Program Scheduling

$XQSB may be used by a device driver to schedule a program, pass it up to five parameters, and also change the terminal logical unit stored in the ID segment.

The calling sequence is:

```
    JSB $XQSB
    DEF <Program Name in 3-word buffer>
    DEF <5-word parameter buffer>
    DEC <new logical unit>
   Return: P+4

    Program not found        A = -1      B = 0
    Program busy             A > 0       B = ID address
    Successful schedule      A = 0       B = ID address
```

If the program is busy, the A-Register will contain the status bits from ID segment word 16.

The parameter address in the 5-word parameter buffer should be direct or indirect to a list of five parameter addresses.

If the logical unit passed is zero, then the terminal LU is not changed in the ID segment.

RTE drivers follow a recommended convention in using the five-word buffer as follows:

Word 1 is the LU of the device from which the schedule attempt was initiated.

Word 2 is an arbitrary value taken from the control request 20B when used to setup the program to schedule on asynchronous interrupt.

Words 3 to 5 are one to three words of device driver status information which may be used by the scheduled program.

Adherence to these recommendations will permit automatic trap handling in BASIC/1000D and BASIC/1000L.

On systems using Security/1000, drivers calling the $XQSB routine for purposes other than scheduling a program on unsolicited interrupt may lock up. (For example, a driver may call an update program everytime it exits.)

Driver lock up can be prevented by using one of the following procedures:

1. The driver can set the value of the executing session number (operating system paramter $XQSN) to zero prior to calling the $XQSB routine. If $XQSB is set to zero, the capability level defaults to the session capability level.

2. If unable to modify the driver source code, use the LINK PC command when linking the program, and specify a value of zero for the RQUSCPLV parameter. This sets the required user capability level to zero.

Mapping Considerations

The operation of a system with mapping (such as RTE-A) requires special procedures for data manipulation as well as DMA configuration. At times, drivers must examine and/or modify an I/O request's buffer. The data buffer may or may not be mapped in. To make mapping as transparent as possible to the driver writer, the operating system includes a set of subroutines that allow the driver to read or write into the data buffer without having to consider mapping. HP strongly advises that all drivers use the subroutines described below.

In A-Series systems:

1. The Operating System is always mapped,
2. the user map (map set 2 or 3) is mapped if data is in the user space,
3. SAM is always mapped (map set 4),
4. the auxiliary map (map set 7) is used so that the user map is not modified, and
5. twenty-four port maps (map sets 8 through 31) are available for DMA, and are dynamically allocated and deallocated between the 48 I/O channels as needed.

The following lists all of the map set assignments:

0	System
1	System/message processor
2	User data
3	User code
4	SAM
5	(reserved)
6	DS
7	Auxiliary
8 - 31	Port maps for DMA access

The L bit referenced below is normally found in DVT15 but can be found in the control word for a multibuffered request. In both cases, the L bit has the same function. Device drivers should set the L bit in multibuffered request control word to indicate which data buffer is being referenced, the original request data buffer, or a data buffer from the driver's area. If the data buffer is in the driver code space (as in a non-partionable driver) or in the driver extension area (as in a partitionable driver), the L bit should be zero. If it is the original request data buffer, this bit should be set the same as the L bit in DVT15.

There are three sets of mapping routines described below. They are $SETM/$READ/$WRIT, $ONER/$ONEW, and $MSALC/$MSRTN. The first set is recommended, because each call to $ONER/$ONEW takes as long as a call to $SETM plus $READ/$WRIT. $SETM must be called before calling $READ/$WRIT, but it must be called only once per entry and it does not have to be called at all during initialization. $MSALC and $MSRTN are used from within a driver for allocating multiple map sets for a driver and also by the I/O system for allocating map sets.

$SETM: Set Up Map Registers

$SETM sets up the map registers for the $READ and the $WRIT subroutines. The $SETM/$READ/$WRIT set of routines is useful if the driver has to manipulate more than one data word between driver entry and driver exit. $SETM does not need to be called if the driver was entered with an initiation directive because the map registers are set up automatically by the system. In all other cases, if a driver is to use $READ or $WRIT, this routine should be called first.

The calling sequence is:

```
    B-Register = DVT address

    JSB $SETM
    Return: P+1  A and B unchanged
```

$READ: Read Data Word/Map Selected

$READ allows the driver to read one word from the data buffer, but assumes that all map registers have been set up prior to this call. This routine, in conjunction with $SETM, should be used if more than one word needs to be read. If a driver has been entered with an "initiate" entrance, this routine can be used without a call to $SETM, because the system has already set up the maps.

The calling sequence is:

```
    B-Register = logical address
                 (base address provided in request plus offset)

    JSB $READ
    DEF (word containing L bit in bit 3)
    Return: P+2     A = Data value;  B, E unchanged
```

$WRIT: Write Data Word/Map Selected

$WRIT is the converse of $READ: it writes one word into the data buffer. If a driver has been entered with an "initiate" entrance, this routine can be used without a call to $SETM, because the has previously set up the maps. In all other cases $SETM must be called to set up the maps.

The calling sequence is:

```
    A-register = Data value to be stored
    B-register = Logical Address
                 (base address provided in request plus offset)

    JSB $WRIT
    DEF (word containing L bit in bit 3)
    Return: P+2   A, B, E unchanged
```

$ONER: Read One Word Without Setup

$ONER allows the driver to read one word from the data buffer. This is useful if the interface driver is resumed, but does not want to go through the overhead of setting up the complete map set. This routine should never be used if the driver has been entered on an Initiate entrance, because the maps have already been set up by the system. $READ should be used in place of this routine.

Calling sequence is as follows:

```
    B-register = logical address of word to read
                 (base address provided in request plus offset)

    JSB $ONER
    DEF (word containing L bit in bit 3)
    DEF (DVT)
    Return: P+3   A = data read;  B, E unchanged
```

$ONEW: Write One Word Without Setup

$ONEW allows the driver to write into the data buffer. Again, this routine should not be called if the driver has been entered on an initiate entrance, because the maps have already been set up by the system. $WRIT should be used in its place.

Calling sequence is as follows:

```
    A-register = Value to be written
    B-register = Logical Address
                 (base address provided in request plus offset)

    JSB $ONEW
    DEF (word containing L bit in bit 3)
    DEF (DVT)
    Return: P+3  A, B, E all unchanged
```

$SETR: Set Port Map

$SETR sets the port map for a request in the DVT. This is useful for setting up the correct port map for the DMA transfer to and from the user buffer. $SETR does not need to be called on request initiation; the driver is entered with the correct port map setting. On any other entry (i.e., asynchronous interrupt which must initiate a DMA request), this routine should be called.

The port map is returned in the A-register, and should be OR'd into the DMA control word of the quad that does the actual data transmission. This routine makes a call to $SELR to obtain the port map number, so both need not be called consecutively (see below).

```
    B-register = DVT address
    JSB $SETR
    Return: P+1   B = starting physical page of transfer
                  A = port map number

    For example: JSB $SETR
                 IOR CNTZ1
```

This would logically OR the port map into the DMA control word.

$SELR: Select Port Map Number

$SELR is used to find out what map set an I/O channel should use for DMA. $SELR will check to see if the I/O is coming from the system or SAM map. If so, and the driver is not going to change the mapping registers, $SELR will return a 0 for the system map or a 4 for the SAM map. Otherwise, it will check to see if a port map has been allocated for this channel. If one has been, that port map number will be returned to the caller. In case a port map needs to be allocated, $MSALC will be called to get one. If no map sets are available, the I/O will be suspended until a map becomes available.

This subroutine must be used when setting up the DMA control register (register 21, see Chapter 9, I/O Card Programming) for the actual data transfer to or from the user's buffer. The number returned should be OR'd into the control word and stored in the self-configuration quad or output directly to register 21. The relocation number in this case is zero. $SELR can be used only if the driver has been entered with an Initiate entrance. In all other cases, $SETR should be used.

The calling sequence is:

```
    A-Register = address of word containing L bit in bit 3
    B-Register = IFT address
    JSB $SELR
    Return: P+1    A = port map number

    For example:   JSB $SELR
                   IOR CNTZ1
```

This would logically OR the port map number into the DMA control word.

$MSALC: Allocate Additional Map Sets

$MSALC allows a driver to allocate additional map sets for setting up multiple DMA transfers. Use $SELR or $SETR to get the first map set, but use $MSALC to allocate any map sets after that. See $SELR, above, for a description of setting the DMA quad.

The calling sequence is:

```
      A-Register = IFT address
      JSB $MSALC
      Return: P+1    A = -1, no port map available
      Return: P+2    A = allocated port map number
```

For example:

```
      LDA IFTA
      JSB $MSALC
      JMP NOMS
      STA MapSetNum
      IOR CNTZ1
```

$MSRTN: Deallocate a Map Set

$MSRTN is used to deallocate a map set that has been allocated using $MSALC.

The calling sequence is:

```
      A-Register = IFT address
      B-Register = Number of port map to return
      JSB $MSRTN
      Return: P+1    Error, port map did not belong to the IFT specified
      Return: P+2    Success
```

For example:

```
      LDA IFTA
      LDB MapSetNum
      JSB $MSRTN
      JMP ERROR
      JMP SUCCESS
```

Chapter 8
Privileged Drivers

A privileged driver is a special interface driver which is permitted to interrupt the operating system and other lower priority privileged drivers.

Privileged drivers have two entry points:

1. ID.yy, the standard entry point for any interface driver.

2. PI.yy, the privileged entry point.

The generator places a JSB to the privileged entry point (through a link word) in the trap cell for the driver's select code. When the interrupt occurs, the driver is entered without the knowledge of the operating system.

For the standard entry point, the driver is written like any other interface driver. This entry is under control of the operating system and used to initiate the request.

On privileged entry, the driver must save the present state of the processor (register values, memory protect fence, etc.) and restore it prior to exit.

Because a privileged driver can interrupt the system, the driver may not use any system routines in its privileged section. If it must post any information in the DVT or IFT, it must have previously-saved pointers to the correct addresses, which can be easily obtained when the request is initiated. The driver must also perform mapping functions, if the data needs to be examined. Because this storage is local to the driver, any privileged driver that handles more than one select code will have to manage separate storage areas for each select code.

Generally privileged drivers must disable interrupts for part of their operations. The interrupts should be enabled whenever possible to permit interrupts by higher-priority privileged select codes to be quickly serviced.

The method of returning from a continuation entry will depend upon whether more entries are expected or the request is done. For a done exit, it will also depend upon whether the system or a program was interrupted. In the latter case, the system may be entered by the driver to complete done processing. Otherwise, done processing is deferred by placing the IFT on a privileged done list for the system to process at the earliest opportunity.

If a program (rather than the system) is interrupted, the driver may enter the system directly to complete done processing. Prior to exit, however, it must set up the register values and the point of suspension of the program that was interrupted; see the table below.

Table 8-1 lists the global values and entry points which must be accessed by the privileged driver to perform the save-and-restore tasks normally performed by the system.

The following program listing shows how the save-and-restore tasks are performed in a privileged driver. The sample driver controls the general purpose interface card. The driver is not a Hewlett-Packard product, and it does not represent current programming standards. It is included only to show how a privileged driver performs tasks normally performed by the system.

Table 8-1. Global Values/Entry Points Needed by a Privileged Driver.

Entry Pt.	Meaning
$SUSP,I	P Register
$A,I	A Register
$B,I	B Register
$CQ	C and Q registers
$EO,I	E and O registers
$X	X register
$Y	Y register
$Z	Z register
$MPTF	Memory Protect Flag (0 = on)
$PDON	Privileged Driver Done Exit (if MPTF off)
$PIMK	Privileged Interrupts Mask value
$Q.PV	Head of privileged drivers done list.
$WMAP,I	Enabled memory maps

```
ASMB,R,L,C
*
      NAM ID.51   781009
*
*  MICROCIRCUIT PRIVILEGED DRIVER FOR HISTOGRAMMING
*
*  DESIGNED TO SERVE AS AN EXAMPLE OF PRIVILEGED DRIVERS
*
      ENT ID.51,PI.51
*
      EXT $A,$B,$EO,$SUSP
      EXT $PIMK,$MPTF,$Q.PV,$WMAP
      EXT,$IF7,$DV15,$DV16,$DV17,$DV18,$DSV19
      EXT,$PDON,.XJCQ,.SIMP,.CIQA,.CZA,$WMAP,$X,$Y,$Z,$CQ
*
      SUP
*
ID.51 NOP           ENTERED FROM IOC
      AND B7
      CPA B1
      JMP INIT      NEW REQUEST INITIATION
*
      CPA B3
      JMP TMOUT     TIMEOUT
*  TREAT AS AN ABORT
      JMP DONEX     TAKE PHY DONE EXIT
      SPC 3
*  REQUEST INITIATION
INIT  LDA $DV16
      STA DVT16     SAVE ADDR OF DVT16
      LDA $DV15,I
      AND B3
      CPA B3        CONTROL REQUEST?
      JMP CNTRL     YES
      CPA B2        WRITE?
      JMP REJCT     YES, REQUEST ERROR
      SKP
*
*  HISTOGRAM REQUESTS HAVE THIS FORMAT:
*               JSB EXEC
*               DEF *+7
*               DEF .1          READ
*               DEF LU          LU OF PRIVILEGED MICROCIRCUIT DVR
*               DEF BUFR        ADDR FOR HISTOGRAM RESULTS
*               DEF LEN         SIZE OF HISTOGRAM BUFFER
*               DEF ADDR        1ST CORE LOCATION TO HISTOGRAM
*               DEF #INWD       # OF WORDS PER HISTOGRAM BUFR CELL
*
*    THE AREA OF CORE HISTOGRAMMED WILL BE FROM (ADDR) TO
*    (ADDR)+(#INWD)*(LEN-1)-1.  THE FIRST WORD OF THE HISTOGRAM
*    BUFFER RECEIVES THE NUMBER OF "HITS" OUTSIDE OF THE ABOVE
*    RANGE.  WHEN ANY CELL REACHES 177777B, IT IS NO LONGER
*    BUMPED, HENCE THIS VALUE REPRESENTS OVERFLOW.
*
      CCA
      ADA $DV17,I   SIZE OF BUFFER-1
      MPY $DV19,I   TIMES # WORDS PER CELL
      SZB,RSS
      CMA,SSA,INA,RSS NEGATE
      JMP REJCT     ERROR IF <0 OR >32767
      STA NRANG     SAVE FOR RANGE CHECKING
      LDA $DV19,I   WDS PER CELL
      STA #WD
      LDA $DV16,I   GET HISTOGRAM BUFR ADDR
      STA BUFAD     SAVE LOCALLY
      LDA $DV18,I   GET CORE ADDRESS
      CMA,INA
      STA NEGAD     - CORE ADDR FOR RANGE CHECK
      DLD #MEAS     NEG # OF HISTOGRAMS (2 WORD)
      DST MEASX
      LDA $IF7      ADDR OF IFT7
      STA IFT7      SAVE LOCALLY
      LDA $IFTX     ADDR OF IFTX
      STA IFTX      SAVE LOCALLY
*
*  NOW START PHOTOREADER TO CAUSE PRIVILEGED INTERRUPTS
*
      ISZ ID.51     TAKE PHYSICAL CONTINUE EXIT
      STC 30B,C
      CLA           NO T.O.
      JMP ID.51,I   EXIT
      SKP
*  THIS IS THE PRIVILEGED INTERRUPT SECTION OF ID.51
PI.51 NOP
      CLC 4         TURN-OFF EVERYBODY
      JSB .SIMP     SAVE
      DEF WMAP      WORKING MAP
      DST ASV       SAVE REGS
      ERA,ALS
      SOC
      INA
      STA EOSV      SAVE E&O
      JSB .CIQA     SAVE Q
      STA QSAV
      JSB .CZA      SAVE Z
      STA ZSAV
      LDA $MPTF     GET MEMORY PROTECT STATE
      STA MPFSV
      ISZ $MPTF     FLAG THAT MEM PROTECT IS OFF
      LIA 2         READ GLOBAL REGISTER
      STA GLOBL
      LIA 4         GET INTERRUPTING S.C.
      OTA 2,C       SET & ENABLE GLOBAL REG
      LDA $PIMK     MASK ALL BUT
      OTA 0            PRIVILEGED INTERRUPTS
      NOP           **TEMP
      STC 4         REENABLE INTERRUPTS
*
*  HISTOGRAMMING UPDATE
      LDA NEGAD
      ADA PI.51     INTERRUPTED LOC-1ST HISTOGRAM LOC
      SSA           OUTSIDE OF RANGE?
      JMP OUTRG     YES
      LDB 0         OFFSET
      ADB NRANG
      SSB,RSS       BEYOND UPPER LIMIT?
      JMP OUTRG     YES
      CLB
      DIV #WD
      ADA BUFAD     ADDRESS HISTOGRAM BUFFER
      INA,RSS
OUTRG LDA BUFAD     OUT-OF-RANGE, USE 1ST LOC
      LDB 0,I       GET CURRENT CONTENTS OF CELL
      INB,SZB       BUMP IT, SKIP IF OVERFLOW
      STB 0,I       NON-OVERFLOW, CELL=CELL+1
*
      ISZ MEASX+1   COUNT TOTAL
      JMP PCONT
      ISZ MEASX     INCR UPPER WORD OF COUNT
      JMP PCONT
      SPC 2
*
*  TOTAL # OF HISTOGRAMS HAS OCCURRED, COMPLETE NOW!
*
      CLC 30B,C     CLEAR CARD
      SPC 2
*  THE BELOW CODE SERVES AS AN EXAMPLE OF HOW PRIVILEGED DRIVERS
*  MAY COMPLETE A REQUEST TO THE OPERATING SYSTEM WITH MINIMUM LATENCY
      SPC 1
      CLC 4         INTERRUPTS OFF
*  UPDATE SYSTEM FLAGS - "T" WOULD BE MEANINGLESS
      LDB IFT7      ADDR OF IFT7
      LDA 1,I       GET IFT WD 7
      AND =B3777    CLEAR BITS 15-11
      STA 1,I       SYS. FLAGS ALL ZERO
      CLA
      STA DVT16,I   POST GOOD COMPLETION
      CPA MPFSV     WAS SYSTEM INTERRUPTED?
      JMP PDNOW     NO, WE CAN ENTER IT NOW
*
*  ENQUEUE THIS IFT ON "$Q.PV" QUEUE OF PRIVILEGED IFTS REQUIRING
*  PHYSICAL DONE PROCESSING SO THAT I/O SYSTEM WILL PERFORM A P.D.
*   FOR THIS IFT RATHER THAN RETURN IMMEDIATELY TO USER PROGRAM
*   WHEN THE CURRENT SYSTEM PROCESS COMPLETES.
*
      LDB IFTX      POINT TO IFT EXTENSION
      LDA $Q.PV     GET CURRENT HEAD OF $Q.PV" QUEUE
      STB $Q.PV     PUT OUR IFT AT HEAD - LIFO
      STA 1,I       LINK TO NEXT GOES IN IFT EXT WD #1
*
RESTR LDA EOSV
      CLO
      SLA,ELA       RESTORE E
      STO           SET O
      LDA GLOBL
      OTA 2,C       RESTORE/ENABLE GLOBAL REG
      LDA MPFSV
      STA $MPTF     RESTORE M.P. FLAG
      LDB ASV+1     RESTORE B REG
      CLA
*
*   NOTE THAT SERVICING OF A TBG TIME TICK MAY
*   DELAY THE INTERRUPTED PRIVILEGED DRIVER.
*   IF THIS IS A PROBLEM, THE PRIVILEGED DRIVER
*   SHOULD RUN WITH INTERRUPTS OFF.  THE TBG TICK
*   WILL THEN BE DELAYED (NOT LOST).
*
      OTA 0         UNMASK ALL INTERRUPTS
      LDA ASV
      STC 4         INTERRUPTS ON
      JSB .XJCQ
      DEF WMAP
      DEF PI.51,I   RETURN TO POINT OF INTERRUPTION
      DEF QSAV
*
*   HERE WHEN MEMORY PROTECT WAS ON SO THAT I/O SYSTEM
*   CAN BE ENTERED DIRECTLY FOR PHYSICAL DONE
*
PDNOW ADB N6        POINT TO IFT WORD 1
      LDA ASV       SAVE MACHINE STATE
      STA $A,I      ON INTERRUPT IN
      LDA ASV+1     ID SEGMENT OF
      STA $B,I      CURRENTLY EXECUTING
*
*   PUT LOCAL STATE WHERE RTE CAN FIND IT
*
      CXA
      STA $X        SAVE X & Y IN
      CYA           USER BASE PAGE
      STA $Y
      LDA QSAV
      STA $CQ
      LDA ZSAV
      STA $Z        SAVE Z-REGISTER FIRST AND RESTORE
      LDA WMAP      SAVE WMAP
      STA $WMAP,I
      LDA EOSV
      STA $EO,I
      LDA PI.51     SET POINT OF
      STA $SUSP,I   PGM SUSPENSION
*  ENTER IOC WITH B REGISTER POINTING TO THE IFT WORD 1
      STC 4         INTERRUPTS ON
      JMP $PDON     PROCESS PHYSICAL DONE NOW!
      SPC 4
PCONT CLC 4         INTERRUPT SYSTEM OFF
      STC 30B,C     RESTART PR
      JMP RESTR     RESTORE REGS & EXIT
      SKP
*
*  HERE FOR CONTROL REQUESTS
*
CNTRL LDA $DV15,I
      AND B7700
      CPA B4000     FUNC 40 TO SET SIZE OF HISTOGRAM
      JMP SETSZ
*
REJCT LDA BN7       =140001 REQUEST ERROR
      JMP DONEX+1
*
SETSZ LDA $DV16,I
      CMA
      LDB $DV17,I   DOUBLE WORD INTEGER
      CMB,INB,SZB,RSS
      INA
      DST #MEAS     - HISTOGRAM CNT (2 WORD)
*
DONEX CLA
      STA DVT16,I   SET ERROR CODE
      CLC 30B,C     ENSURE PR DISABLED
      CLA
      JMP ID.51,I   PHYSICAL DONE EXIT
      SPC 3
*  TIME-OUT
TMOUT LDA B3
      JMP DONEX+1   RETURN ERROR 3
      SPC 3
*  DATA AREA
IFT7  NOP
IFTX  NOP
DVT16 NOP
BUFAD NOP
NEGAD NOP
NRANG NOP
#WD   NOP
```

```
*
WMAP  NOP
EOSV  NOP
QSAV  NOP
ZSAV  NOP
GLOBL NOP
MPFSV NOP
*
#MEAS DEC 0,0
MEASX DEC 0,0
*
B1    OCT 1
B2    OCT 2
B3    OCT 3
B7    OCT 7
B4000 OCT 4000
B7700 OCT 7700
*
N6    DEC -6
BN7   OCT 140001
*
ASV   BSS 2
*
      END
```

Chapter 9
I/O Card Programming

This chapter briefly describes how the system performs I/O. For more detailed information, refer to the Operating & Reference Manual for the A-Series processor.

Two kinds of I/O programming are possible: interrupt per word or byte and interrupt per block. The latter uses DMA (direct memory access).

I/O instructions are executed by an I/O microprocessor chip common to every I/O card. The central processor and the I/O chip communicate along the backplane bus. When communication takes place, the I/O chip and the central processor operate as a single computer to process I/O transfers through an I/O channel.

The two-digit octal select code represents the address of the I/O interface card on the backplane and is the basis for linking the main processor with a particular I/O card. Bits 5-0 of an I/O instruction may reference either the select code or a register on the I/O chip, depending on the state of the global register and the actual value in bits 5-0.

Select codes 20B through 77B are available to I/O drivers. The choice of which select code to use (controlled by jumper on the interface) depends on the following:

1. The privileged interrupt mask controls a group of four select codes with a single bit. Therefore, the generator will report an error if a normal and privileged driver are assigned to the same bit in the mask.

2. Conventions established for use at the local site.

Each interface card contains the I/O chip common to all I/O cards, and card logic unique to the function of the card. The I/O driver communicates with the card logic by accessing registers on the I/O chip. The chip manages the card logic to enable data transfers in the DMA mode or in the interrupt per byte or word mode.

The select code field in an I/O instruction can specify the chip register. Each register has associated with it a control bit and a flag bit (these are 1-bit registers) to manage the direction of data flow. Generally, the control bit is set to indicate that the driver is ready, and the flag is set to indicate that the device is ready. Usually, the flag is cleared by the driver when the transfer is initiated: when the device finishes, the flag is set by the device (or the I/O chip) to generate an interrupt. The flag may also be set by the driver to abort or suspend a transfer.

The registers are numbered 0 through 77 octal, but only the following registers are of interest to the I/O driver:

1. Global Register: register 02. When the global register is enabled, its contents specify an interface card which is to process I/O instructions whose select code is in the range 20B-77B.

2. Virtual Control Panel: register 24. This register is used to indicate the use of the card by the Virtual Control Panel code.

3. Card Registers: registers 30, 31, 32. The card registers control the card logic which is unique to the function of the card.

4. DMA Registers: registers 20, 21, 22, 23. These registers are used to manage block transfers to and from memory.

The I/O chip will always recognize select codes 02 and 03, regardless of the state of the global register. In addition, with the global register disabled, the chip will recognize instructions addressed to its own select code.

The system's standard interface drivers (i.e. non-privileged drivers) are always entered by the system with the global register set and enabled for the select code taken from the IFT. Therefore, those drivers need not concern themselves with register 02.

The card registers (30, 31, 32) are accessed in the same manner for each card. The contents and meaning of these registers is unique to the card function.

Handling of the DMA registers is nearly the same for every card. The differences are for the convenience of the driver, rather than required by the card.

The card registers and the DMA registers are described separately in the following sections.

The Global Register

The global register is 6 bits wide and is designed to contain a select code. The register is loaded and read by the instructions:

OTA/B 2	Load Global Register from A/B
LIA/B 2	Read Global Register into A/B
LIA/B 2,C	Read and clear the flag
MIA/B 2	Merge with A/B Register
MIA/B,2,C	Merge and clear the flag

The value loaded into the global register must be in the range 20B-74B; else the interface card will go into a diagnostic mode.

The global register is enabled/disabled by:

```
    CLF 2          Enable Global Register
    STF 2          Disable Global Register
```

and tested with:

```
    SFS 2          Skip if the flag set
    SFC 2          Skip if the flag clear
```

All I/O chips recognize select code 2, regardless of the state of register 02. When register 02 is disabled, however, the I/O chip will recognize only register 02 and the register corresponding to its own select code.

When the global register is enabled, the select code is used to indicate a register in the range 20B-74B. However, not all of these registers have defined usage.

Hewlett-Packard interface cards and RTE-A interface drivers are designed to be used only with the global register enabled.

Virtual Control Panel Register

Register 24 is used by the Virtual Control Panel to indicate that it has used that I/O card. This register will be set equal to minus one (-1) prior to exit from the VCP program. This value signals the driver to restart any request which may have been aborted as a result of the VCP operation.

The control and flag bits are set on the I/O cards used. Thus, upon return to the operating system, an interrupt will occur. If the select code is used by a driver in the system, then a continue entry will be made into the interface driver to service the interrupt. If no driver uses the select code, the interrupt is ignored by the system.

Drivers which use register 24 include terminal drivers that can process the keyboard used by the VCP program and any boot devices which may be referenced by the VCP program. For example, since a boot may occur over the network interface, then the driver for the network interface card should use register 24.

Below is an example of the procedure the driver should follow to use register 24:

```
      LIA 24B
CONT  SZA,RSS      REMOTE CONTROL INTERRUPT?
      JMP CONT1    NO. CONTINUE PROCESSING
      CLA          YES.
      OTA 24B      CLEAR INDICATOR REGISTER
*
* PERFORM ESSENTIALLY THE SAME PROCESS AS IF
* A POWER-FAIL RESTART. EXAMPLE BELOW.
*
PWRFL LDB $IF6,I   GET AVAILABILITY
      SSB,RSS      BUSY?
      JMP BRK      NO. RESET ANY ASYNCH INTERRUPTS EXPECTED.
      LDA REDO     RESTART REQUEST IN PROGRESS
      STA $DV16,I  DON'T DOWN,DON'T FLUSH,NO ERR MESS
      CLC ZIB,C
      CLC 23B,C    TERM ANY DMA SO NO CONFLICT ON REENTRY
      JSB STAT
      CLA          SYSTEM FLAGS = 0
      JMP ID.XX,I  "DONE" EXIT.
*
REDO  OCT 100077   "D" BIT + ERROR CODE 63
*
CONT1 EQU *
* NORMAL PROCESSING CONTINUE HERE
```

Prior to exit, the driver should always clear register 24. To accomplish this:

```
CLA
OTA 24B
```

Card Registers

The interface card registers, 30, 31, and 32, are accessed only with the global register enabled. The instruction set is given below with XX representing the register number 30 to 32.

LIA/B XX	Move card register to A/B register
LIA/B XX,C	Move and clear the flag
MIA/B XX	Merge card register into A/B register
MIA/B XX,C	Merge and clear the flag
SFS XX	Skip if the flag set
SFC XX	Skip if the flag clear
STC XX	Set device control
STC XX,C	Set control and clear flag
STF XX	Set the flag
CLF XX	Clear the flag

Register 30 stores data. An OTA sends data to the card; an LIA removes a data word from the card.

Register 31 is for card control. An OTA sends a word to the card which "configures" it. Configuration affects the way the card handles data and is analogous to setting jumpers on the interface card.

An LIA 32 instruction reads the card status, which may include device status as well, depending on the card.

Register 32 is not used on all I/O cards. Where used, it is specific to the card. The following examples illustrate how to use the card registers.

USING THE CARD REGISTERS FOR INPUT

The operations below assume that the global register is set up and enabled.

```
LDA CNTRL     Get control word
OTA 31B       Output to card
STC 30B,C     Start device
...
```

The FLAG on the data register (30) is set when the device is ready with data. In addition, FLAG 30 is the flag for the entire interface card and may, therefore, generate interrupts, if they are enabled. FLAG 30 may also be tested under program control as follows:

```
SFS 30B       Wait for data flag ready
JMP *-1
LIA 30B       Now get data
```

The FLAG remains set until reset under program control, which is normally done when a new operation is started. To intiate another input:

```
STC 30B,C
```

When the final value has been read, the interface should be set to a known state by:

```
CLC 30B,C
```

which clears the card control and flag.

USING THE CARD REGISTERS FOR OUTPUT

The operations below assume that the global register has been set up and enabled.

```
    LDA CNTRL     Get control word
    OTA 31B       Output to card
    LDA DATA,I    Get first data word
    OTA 30B       Send to card
    STC 30B,C     Start the card going
    ...
```

When the data transfer is complete, FLAG 30 will be set and may generate an interrupt or be tested in the same manner as discussed under INPUT. To start the next transfer:

```
    ISZ DATA      Increment the data pointer
    LDA DATA,I    Get next data word
    OTA 30B       Send to the card
    STC 30B,C     Restart the device
    ...
```

Again, at completion the card should be reset as follows:

```
    CLC 30B,C     Clear card control and flag bits
```

DMA Registers

Incorporated into every I/O chip is the ability to transfer data directly to or from memory. All necessary control logic and registers are contained in the I/O chip to supervise the memory transaction. The I/O chip and logic circuits on the interface card interact to manage the flow of data and control signals.

There are four DMA control registers on the I/O chip:

Reg#	Purpose
20	DMA Self-Configuration Register
21	DMA Control Register
22	Address Register
23	Data Count Register

The self-configuration feature permits DMA transfers to be chained together. The individual transfers are described by triplets (or quadruplets) in processor memory. When using the DMA chaining feature, only the address of the first chain needs to be given to the I/O chip. When one transfer completes, the next is initiated automatically with very little overhead.

DMA Initiation

The I/O sequence required to initiate a non-chained DMA transfer is as follows:

```
        LDB $DV1      SET THE PORT MAP
        JSB $SETR
        IOR CNTL      GET DMA CONTROL WORD
        OTA 21B       OUTPUT TO CONTROL REGISTER
        LDA ADDR      GET ADDRESS OF MEMORY BLOCK
        OTA 22B       OUTPUT TO DMA ADDRESS REGISTER
        LDA CNT       GET DATA COUNT
        OTA 23B       OUTPUT TO DATA COUNT REGISTER
        STC 21B,C     START DMA AND CLEAR INTERRUPT FLAG
        ...
        ...
  CNTL  BSS 1         SET UP BY PROGRAM TO DESCRIBE TRANSFER
  ADDR  DEF BUFR      POINTS TO STORAGE
  CNT   BSS 1         DATA COUNT STORED HERE AS NEGATIVE VALUE
```

The data count is initialized to the negative of the word count or the byte count, according to bit 13 in the control word.

The sequence needed to initialize a chained transfer is even simpler:

```
  LDA PNTR     GET ADDRESS OF CHAIN
  OTA 20B      TELL IT TO I/O CHIP
  STC 20B,C    CLEAR INTERRUPT FLAG AND START CHAIN
```

A sample chain is given below. The example also illustrates the bits in the DMA control word.

```
  CONT  EQU 100000B   CONTINUE SELF-CONFIGURATION CHAIN
  DEVCM EQU 040000B   ISSUE DEV COM PULSE AFTER EA WD/BYTE
  BYTE  EQU 020000B   DATA COUNT IS IN BYTES
  RES   EQU 010000B   OVERWRITE DATA COUNT WITH RESIDUE AT END
  CINT  EQU 004000B   INHIBIT DMA INTERRUPT FLAG
  REM   EQU 002000B   USE REMOTE MEMORY
  FOUR  EQU 001000B   THIS LINK IS A QUADRUPLET
  AUTO  EQU 000400B   DON'T WAIT FOR SRQ FROM DEVICE
  IN    EQU 000200B   TRANSFER IS TO MEMORY FROM DEVICE
  RELOC ....          MUST BE OR'ED INTO CONTROL WORD BEFORE
  *                       STARTING THE TRANSFER
  *
  PNTR  DEF BUFR
  *
```

```
BUFR  ABS CONT+AUTO+RELOC        DMA CONTROL: OUTPUT TRIPLET
      DEF DATA       ADDRESS OF MEMORY BLOCK
CNT1  DEC -10        NEG OF WORD COUNT IN DATA
*
      ABS CONT+AUTO+IN+RELOC  DMA CONTROL: INPUT TRIPLET
      DEF INPT       ADDRESS OF INPUT BUFFER
CNT2  DEC -10        NEG # WORDS IN INPUT BUFFER
*
      ABS BYTE+AUTO+FOUR+RELOC  DMA CONTROL:LAST LINK IS QUADRUP.
      DEF CTRL       THIS IS A CONTROL WORD FOR I/O CARD
      DEF DONE       ADDRESS OF LAST BLOCK
      DEF CNT3       BYTE COUNT OF BLOCK AT DONE BUFFER
*
DATA  BSS 10         OUTPUT DATA BUFFER
INPT  BSS 15         INPUT DATA BUFFER
DONE  BSS 5          FINAL BUFFER IN CHAIN
      ...
```

The chain is "self-configuring" because the I/O chip takes over loading its registers 21, 22 and 23 from the consecutive memory locations beginning at the pointer which is put in register 20. As each memory location is accessed, the value of register 20 is incremented by the I/O chip. The new value in register 20 is used as the address of the next memory read.

If the "FOUR" bit is set in the DMA control word, then the second word in the link is loaded into chip register 31, the card control word. Subsequent words are loaded into registers 21, 22, and 23 - the same as for the triplet.

The self-configuration timing is variable, according to whether the DMA interrupt flag is on or off, and whether this is an initial configuration (top of chain) or a reconfiguration (subsequent link in chain). The hardware manual should be consulted for actual times. However, the self-configuration will always execute faster than the equivalent loading of registers 21, 22 and 23 by the driver itself.

DMA Termination

A DMA transfer can terminate from several causes:

1. The data count goes from -1 to 0. This means that the I/O chip has completed the number of I/O cycles specified in register 23. It does not mean all cyles resulted in a successful memory access. For example, if several high-speed synchronous devices are competing for memory, a lower priority interface may experience a DMA overrun.

2. End-of-transmission. This is determined by the individual interface card, which may recognize a record terminator. For example, an input from a terminal may complete with a carriage return--regardless of data count specified in register 23. The transfer will never exceed the DMA count.

3. Memory parity error during DMA input transfer.

In addition, the I/O driver can programmatically suspend or abort a DMA operation. This may be desirable when the driver is called upon to perform the abort function. This feature will be described further in the section on DMA Flags.

The "residue" (DMA count at completion) may be read from the chip register 23 (LIA/B 23). In addition, if the RES bit is set in the DMA control word, the residue will be written into the same word from which data count was taken (chained operation only). Assuming no parity error or device error, the residue can be used to determine the actual number of words/bytes transferred on output or input.

DMA Control and Flag Bits

The I/O instructions that permit the driver to manage the control and flag bits for each of registers 20 through 23 follow, with descriptions of their functions.

Reg	Instr	Meaning
20	STC	Enable DMA self-configuration logic
	CLC	Suspend self-configuration logic
	STF	Set self-configuration flag
	CLF	Clear self-configuration flag
	SFS	Test flag
21	STC	Enable non-chained transfer
	CLC	Suspend current DMA operation
	STF	Set DMA flag
	CLF	Clear DMA flag
	SFS	Test if DMA flag set (operation complete)
22	STC	Not implemented; NOP
	CLC	Abort current DMA operation, proceed to next self-configured operation
	STF	Set DMA parity error flag
	CLF	Clear DMA parity error flag
	SFS	Test if DMA parity error
23	STC	Not implemented; NOP
	CLC	Abort self-configuration and any transfer in progress
	STF	Set all three flags: 20, 21 and 22
	CLF	Clear all three flags: 20, 21 and 22
	SFS	Test if any of three flags set (20, 21 and 22)

When an operation is suspended, it "pauses," and may be restarted with the appropriate STC instruction. If the device is synchronous, however, the effect of the pause may be lost data. This is commonly called a "DMA overrun," which means that the computer did not process the transfer before the next piece of information was presented. DMA overruns may also occur if several DMA transfers are in progress and high-prioity select codes hold off transfers from low priority synchronous devices.

At the end of a DMA transfer, up to 5 flags may be set as follows:

Flag 20: Set upon completion of the last link in a chained transfer. Flag 20 is set if the residue has gone to zero. The occurrence of flag 20 will also set flag 21.

Flag 21: Set upon completion of a block transfer except when using self-configuration. Like flag 20, it means that the residue has gone to zero. If self-configuration is in effect, the flag 21 is set by the occurrence of flag 20.

Flag 22: Set if a memory parity error occurred on DMA output from memory. Unlike flags 20 and 21, flag 22 is not inhibited by the CINT bit in the DMA control

Flag 23: This is an inclusive OR of flags 20, 21 and 22.

Flag 24: DMA operating flag. Set if a DMA operation is in progress.

Flag 30: Set at completion of DMA transfer only if this is a feature of the card. Some cards use this flag only for non-DMA transfers.

HEWLETT PACKARD

MANUAL PART NO. 92077-90019
Printed in U.S.A. June 1983
E0683

HEWLETT-PACKARD COMPANY
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014