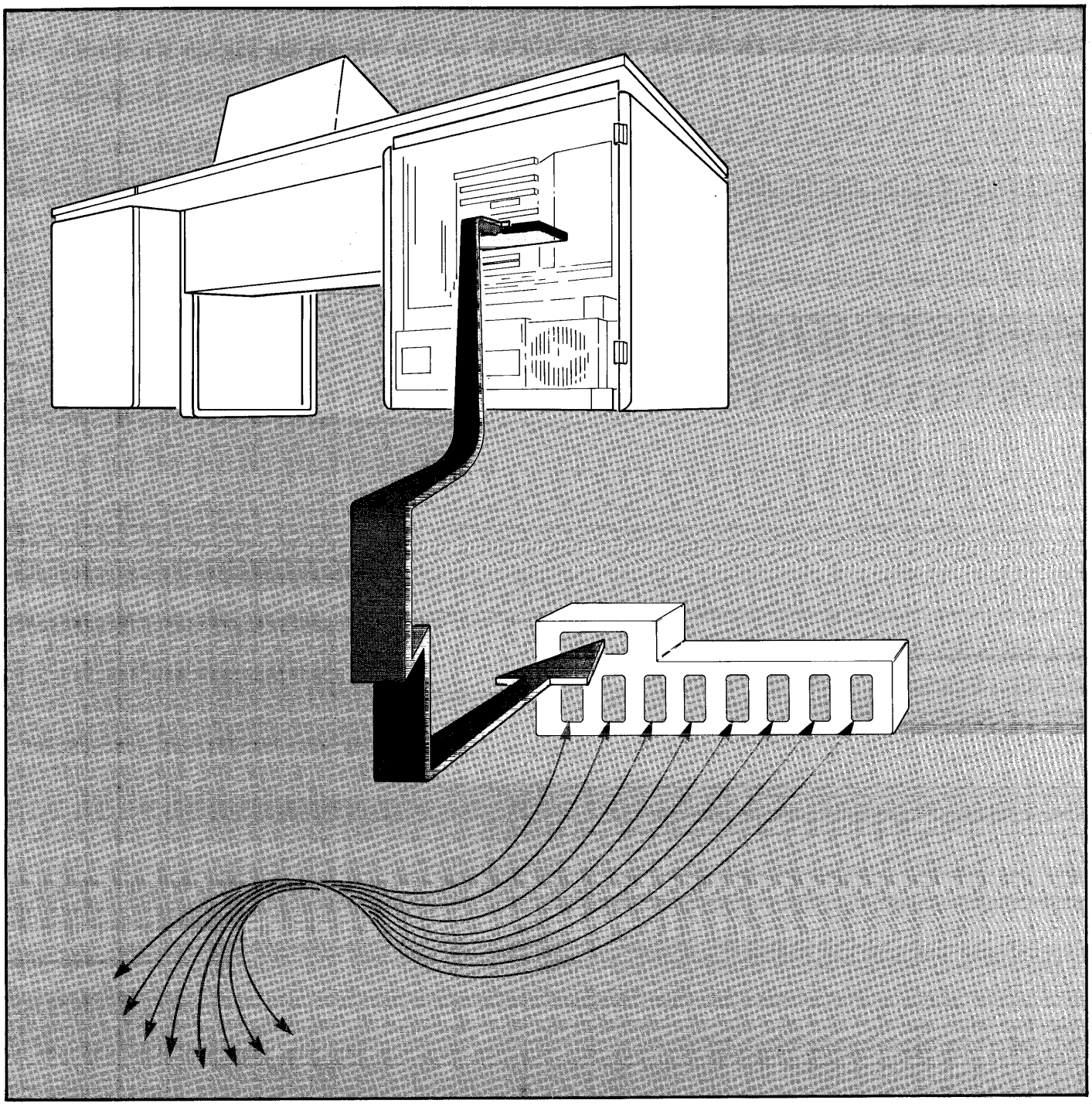
HEWLETT PACKARD

HP 12792A

8-Channel Asynchronous Multiplexer Subsystem User's Manual

for HP 1000 M/E/F-Series Computers



HP 12792A
8-Channel Asynchronous Multiplexer Subsystem

User's Manual

HEWLETT PACKARD

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

Library Index No.
MUX.310.12792-90002

MANUAL PART NO. 12792-90002
Printed in U.S.A. September 1980

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 the latest replacement pages and write-in instructions to be merged into the manual, including an updated copy of this Printing History page.

To replenish stock, this manual will be reprinted as necessary. Each such reprinting 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.

To determine the specific manual edition and update which is compatible with your current software revision code, refer to the appropriate Software Numbering Catalog.

First Edition ..... Sept. 1980

NOTICE

The information contained in this document is subject to change without notice.

HEWLETT-PACKARD MAKES NO WARRANTY OF ANY KIND WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

This document contains proprietary information which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another program language without the prior written consent of Hewlett-Packard Company.

Copyright © 1980 by HEWLETT-PACKARD COMPANY

Preface

This manual is designed as a technical reference for the Hewlett-Packard 12792A Eight Channel Asynchronous Multiplexer Subsystem for M/E/F-Series HP 1000 Computers. The Multiplexer interface is an efficient, high performance interface for multiplexed terminal/device applications. It enhances terminal communications by offering a low cost/channel, high speed, high performance alternative to other point-to-point interface offerings.

This manual is intended for users knowledgeable in FORTRAN and the RTE-MIII or RTE-IVB operating system. The following is a brief description of each chapter's content.

The OVERVIEW provides general information concerning the multiplexer subsystem and its interrelationships with HP 1000 products.

The USER INTERFACE outlines the control functions used to perform data transfers to and from external devices, in addition to the control functions necessary to accomplish I/O control. Examples are provided for each control request.

Using the features of the multiplexer and error handling/recovery is discussed in Chapter 3, USING THE MULTIPLEXER.

The DEVICE DRIVER chapter is directed at the advanced programmer who is experienced with Assembly language. It provides the user with a tutorial on device driver writing.

The DEVICE SPECIFIC CONSIDERATIONS chapter explains the interfacing requirements for using a non-HP device in the multiplexer subsystem.

Table of Contents

Chapter 4 Device Driver Writing

Chapter 5 Device-Specific Considerations

Appendix A Device Equipment Table

Appendix B Device Driver Interfaces

Appendix C Glossary

List of Illustrations

List of Tables

Chapter 1
Overview

The Hewlett Packard 12792A Eight Channel Multiplexer Subsystem and its accessory, the HP 12828A accessory panel, provide communication to the computer through a microprocessor-based interface. The multiplexer significantly off-loads routine communication management overhead. The Multiplexer is used to route data from one of eight I/O devices to a common destination. One possible MUX configuration is shown in Figure 1-1.

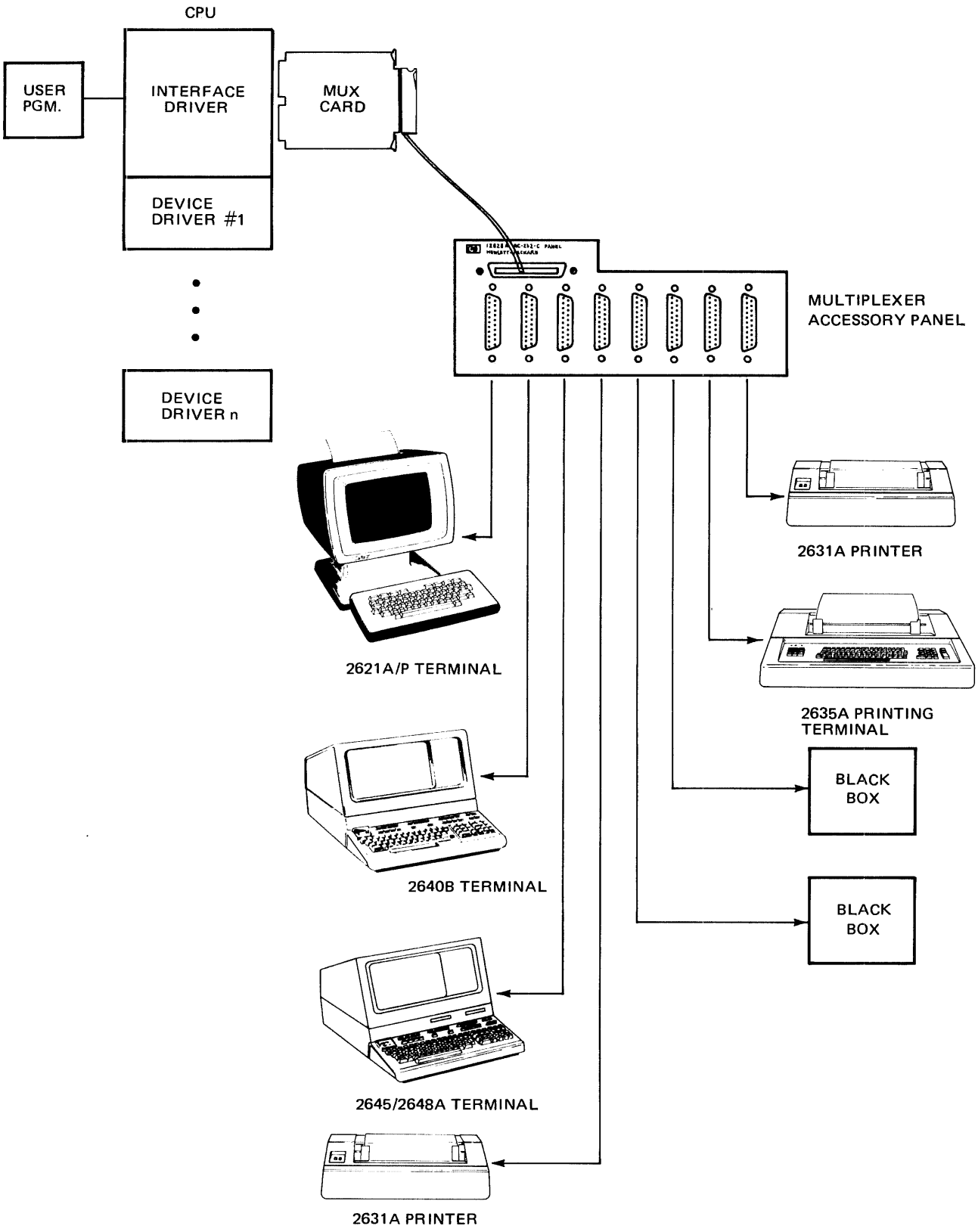


Figure 1-1. Possible MUX Configuration.

7700-558

Flexibility is a key feature of this product in that the users can connect devices such as terminals, printers, and "black boxes" to the MUX, provided the device meets RS-423-A or RS-232-C requirements. Furthermore, users can write device drivers for these devices attached to the MUX to handle any special control required by the device.

Device drivers are simply subroutines of the interface driver which are used to modify user requests and make them compatible with a specific device. The interface driver, on the other hand, is basically responsible for the transfer of information between the user programs, the appropriate device driver, and the interface card.

Each channel of the Multiplexer has a device driver associated with it. The device driver performs the device specific formatting of data, specifically, the inclusion of control characters for device requirements. When an EXEC call is issued from the user program, the interface driver passes it to the device driver. At this point the device driver can break up the user request into a series of interface requests. For example, the device driver can instruct the interface driver to wait for a buffer and inform the interface driver of the buffer's destination. The information is taken from the appropriate channel buffer and sent to the destination indicated by the device driver.

Up to 14 device drivers may be used, each of which can be associated with one or more devices attached to one of the MUX cards in the system. HP supplies two device drivers with the 12792A product -- DDV05 (26XX terminal screen mode device driver) and DDV12 (2631/2635/7310 line printer device driver).

Each Multiplexer interface card contains 16K bytes of Random Access Memory, of which 8K bytes are allocated for channel buffers. This 8K portion of memory is divided so that each channel contains four 254 byte buffers, two for transmission and two for reception. Each interface card provides two on-board programmable baud rate generators which control channel transmission speeds ranging from 50 to 19.2K baud. The total aggregate throughput must not exceed 78.6K baud. This card may be inserted anywhere in the backplane of the CPU, unless there is a privileged interrupt fence. In this case the interface card should be inserted above the fence. The maximum number of physical devices which will be supported in the Multiplexer Subsystem is 61 (RTE-IVB) or 62 (RTE-MIII). There are 63 available EQT's in the system but one EQT should be reserved for the system console and one for the disc. The Multiplexer does not offer system console support.

To increase the throughput of the interface card Direct Memory Access is used. If the terminal/device channel is unable to obtain a DMA channel the device driver is placed into a loop to perform read/write functions on a word-by-word basis.

The HP 12828A multiplexer panel contains eight RS-232-C ports. The standard accessory panel is hardwired connecting port 0 to one baud rate generator, and ports one through seven to another baud rate generator.

The Multiplexer Accessory Panel is connected to the interface card and can be located locally at the CPU (with the standard cable) or up to 91 metres (300 feet) away from the CPU with custom cabling. RS-232-C compatible devices can then be connected to the Multiplexer panel by cables less than 39.7 metres (50 feet) in length.

The MUX will passively support asynchronous, full duplex modems. Passive support means the MUX does not recognize or supply any modem control or status lines. The RS-232-C connectors on the HP 12828A have modem "Clear to Send" connected back to "Request to Send" and "Carrier Detect" wired back to "Data Terminal Ready." ("Data Set Ready" is pulled to +12v through a 1K ohm resistor for those terminals that require it.) Use of the HP 12828A or connectors similar to these allow the passive support of modems. The user should be aware that "Line Loss" or other "Modem Not Ready" problems cannot be detected by the MUX or user software through the MUX. Modem connect or disconnect sequences cannot be requested through the MUX.

Chapter 2
User Interface

General Considerations

This section describes the driver as seen by the user. Standard I/O EXEC calls are used to transfer data to and from external I/O devices in addition to performing various I/O control operations. Input, output and control requests to the multiplexer are generally in the form of RTE EXEC calls while control requests can be initiated either from EXEC calls or by using the file manager CN command. EXEC calls can be made from Assembly Language programs or from higher level languages such as FORTRAN and PASCAL.

Request Code

Parameter ICODE identifies the type of EXEC call request. There are eight types of EXEC calls described in this manual; four normal I/O EXEC calls and four Class I/O EXEC calls.

STANDARD EXEC CODE PARAMETERS

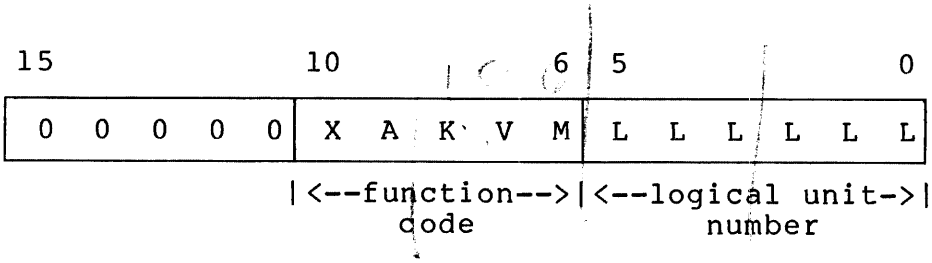
ICODE = 1	READ REQUEST
ICODE = 2	WRITE REQUEST
ICODE = 3	CONTROL REQUEST
ICODE = 13	I/O STATUS REQUEST

CLASS I/O EXEC CODE PARAMETERS

ICODE = 17	READ REQUEST
ICODE = 18	WRITE REQUEST
ICODE = 20	WRITE/READ REQUEST
ICODE = 19	CONTROL REQUEST

Control Word

Control word (ICNWD) contains a five bit function code and the logical unit number (LU) of the device to which the user request is directed. It is structured as follows:



Function Code

The octal value of the required function code is provided for each of the request descriptions in the following sections. The user may choose any of the methods described in this chapter to set the value of bits 10 through 6 of control word ICNWD. For example, if the function code value is octal 6, add 600B to the value of the LU number.

The MUX driver examines the value of the function code to determine the action taken by the interface driver in the processing of I/O or device control. The requests will vary depending on the function codes described below. These function codes are used whenever an EXEC 1 (Read), EXEC 2 (Write), or EXEC 3 (Control) call is made, although the bit meanings differ between read and write requests. Refer to Figure 2-1 for ICNWD bit names.

BIT 10 (X BIT)	Transparent Mode Bit; 0 - DISABLED 1 - ENABLED
BIT 9 (A BIT)	Special buffer control bit; 0 - DISABLED 1 - ENABLED
BIT 8 (K BIT)	Echo Bit; 0 - DISABLE 1 - ENABLE
BIT 7 (V BIT)	Honesty Bit; 0 - DISABLE 1 - ENABLE
BIT 6 (M BIT)	Binary Mode Bit; 0 - DISABLE 1 - ENABLE

Figure 2-1. ICNWD Function Code Bits.

Logical Unit Number

The logical unit (LU) number is the system address for the I/O device to which the user is directing a request. The user's system generation listing enumerates the LU numbers of all I/O devices generated into the system.

For example, if the LU number is decimal 10 and the function is octal 6, the value of ICNWD can be computed:

```
ICNWD = 10 + 600B
```

The B suffix is used to identify "600" as an octal number.

I/O Requests

I/O requests are handled in a variety of ways. These requests are processed differently depending on the value of the control word. Three basic input functions are controlled by the value of ICNWD.

- editing
- echoing
- terminators

When editing is enabled, the control word bits 6 and 10 are set to zero. If a delete key (rub out key) is struck, the contents of the user's receiving buffer is erased. The backspace key deletes only the last character entered, if any. If editing is disabled, the delete and backspace keys would enter a 177B or 12B into the user's on-board buffer.

If a user is inputting data with editing enabled the interface card will accept the data into one of the channel's input buffers. The multiplexer handles all the edits. User input and editing can continue until the card's buffer is full or a valid terminator is detected. At this point the on-board buffer is off loaded to the CPU. Once the buffer has been moved it cannot be edited. For this reason the user cannot backspace or delete past a 254 byte boundary.

In general, information is supplied to the Multiplexer card in a character mode format and echoed back to the user for visual inspection of the buffer. This is not always desirable; the echo feature may be suppressed for passwords and special requests.

The Multiplexer card must be able to detect an end-of-record or valid terminator when it is encountered. Figure 2-2 lists the valid terminators used to signal the card to interrupt the CPU and transfer data to the user's buffer.

Figure 2-3 describes the control word bit combinations used to determine the interface driver's action on a read request. It is here that valid terminators, as well as input editing and echoing are specified. Figure 2-4 describes the valid buffer transfer terminators.

COMMON NAME		OCTAL VALUE OF RIGHT BYTE
carriage return	CR	000015
device control	DC2	000022
record separator	RS	000036
end of transmission	EOT	000004

Figure 2-2. Valid Terminators

ICNWD BITS

10	9	8	7	6	ACTION TAKEN FOR READ REQUEST
0	0	0	X	0	editing enabled echo disabled CR is a valid buffer transfer terminator CNTRL D results in an EOT status condition and a zero length transmission log (zero length buffer)
0	0	1	X	0	input editing enabled echo enabled CR is a valid transfer terminator CNTRL D results in an EOT status condition and a zero length transmission log (zero length buffer)
0	0	0	X	1	input editing disabled echo disabled data transfer terminates only when the user buffer is full
0	0	1	X	1	input editing disabled echo enabled data transfer terminates only when the user buffer is full
1	0	0	X	0	input editing disabled echo disabled CR is a valid transfer terminator
1	0	1	X	0	input editing disabled echo enabled CR is a valid transfer terminator
*	1	*	X	*	special buffer transfer same as the transfer with bit 9 (special function bit) set to zero but data resident in the card's buffer that exceeds the end of the user buffer is not destroyed. It may be accessed in subsequent buffer transfers. echo, edit, etc. are defined by bits 6, 8, and 10 above. X = don't care condition

Figure 2-3. Read Request Function Codes.

ICNWD BITS

10	9	8	7	6	WRITE REQUESTS
0	X	X	X	0	An ASCII write request will have CR/LF appended to the buffer if the last character in the buffer is not "_". If the underscore, 00137B, is appended to the user's buffer it will not be printed.
0	X	X	X	1	the entire buffer is transmitted as is, no characters are appended to the user's buffer.
1	X	X	X	0	
					X = don't care condition

Figure 2-4. Write Request Function Codes.

There are some areas of I/O request handling that require special mention:

Zero-length keyboard entries will not be ignored by the interface driver. A carriage return without data is a zero length record.

Control function 30 configures the Multiplexer card to conduct I/O transfers in a specified character format. The terminal must be configured accordingly for successful I/O processing to occur.

Standard I/O EXEC Calling Sequences

The following sections show the general formats used for making EXEC calls from RTE Assembly, RTE FORTRAN IV and PASCAL/1000 programs. In the following examples, the only names that must be used as given are EXEC and ABREG. Other parameters used such as ICODE, ICNWD, and IPARM are simply mnemonics used in this manual.

EXEC CALLS from Assembly Language

The Assembly language calling sequence for EXEC calls is as follows:

```
                .
                .
                .
                EXT EXEC        Declare EXEC as an external
                .
                .
                .
                JSB EXEC        Transfer control to RTE
                DEF RTN         Return address
                DEF ICODE       Request code
                DEF ICNWD       Control word
                DEF IPR1        Parameter 1, optional
                DEF IPR2        Parameter 2, optional
      RTN                       Return Point
      *                            A register contains I/O status
      *                            B register contains the length of the
      *                              transmission log
                  .
                  .
                  .
      ICODE     DEC 1           Request code word (READ)
      ICNWD     OCT CNWD        Control word. Control function
      *                         + LU number assigned to the port.
                  .
                  .
                  .
      IPR1      OCT pr1         Use depends on
      IPR2      OCT pr2         type of call
```

The return point, labeled RTN, must follow the DEF of the last parameter used. EXEC uses this address to calculate the number of parameters passed for those calls that have optional parameters.

EXEC CALLS from FORTRAN

To call EXEC as a subroutine from RTE FORTRAN IV, use the following calling sequence:

```
      .
      .
      .
ICODE = 3
ICNWD = 600B + LU
      .
      .
      .
CALL EXEC(ICODE,ICNWD,IPARM1,IPARM2)
CALL ABREG(IA,IB)
      .
      .
      .
```

EXEC can also be called as a function from RTE FORTRAN IV, using the following calling sequence:

```
      .
      .
      .
DIMENSION IREG(2)
EQUIVALENCE (REG,IREG),(IA,IREG),(IB,IREG(2))
      .
      .
      .
REG = EXEC(ICODE,ICNWD,IPARM1,IPARM2)
      .
      .
      .
```

The two different methods of calling EXEC from FORTRAN illustrate the two ways of obtaining the A- and B-Register values from the EXEC call.

In the following sections, the examples used to illustrate the calls are written using the CALL EXEC(...) method, with an example showing how to make the control request from the File Manager using the CN command.

EXEC CALLS from Pascal

An EXEC call may be coded in PASCAL/1000 either as a procedure or as a function. If it is coded as a function, the return value type must be a one-word type to return the values of both the A- and B-Registers.

The PASCAL/1000 compiler does not treat an EXEC call in any special manner. Therefore, it is possible to call EXEC directly if an external declaration has been made with a set of formal parameters.

If the $HEAP 2$ compiler option is used, then the HEAPPARMS option must be OFF for EXEC external declarations with VAR parameters.

Example:

```
program muxex;

const                                          { EXEC 3 , a control request  }
      lu_control_request_code = 3 ;            { is placed on LU 19.  The 16 }
                                               { bit control word is stripped}
      shift_left_six_bits     = 64 ;           { of bits 15 -11, so that the }
                                               { function code, bits 10-6,   }
      lu_number               = 19 ;           { can be examined.  In this   }
                                               { case the function code 6    }
      function_code           = 6 ;            { is placing a status request }
                                               { to LU 19                    }
type
     int = -32768..32767;                      { single-word integer type    }
     bit = 0..1;                               { single-bit type             }
     bit_def = packed record                   { bit data type               }
               bit_15 : 0..1;                  { This allows the user to     }
               bit_14 : 0..1;                  { access each bit field       }
               bit_13 : 0..1;                  { individually.               }
               bit_12 : 0..1;
               bit_11 : 0..1;
               bit_10 : 0..1;
               bit_9  : 0..1;
               bit_8  : 0..1;
               bit_7  : 0..1;
               bit_6  : 0..1;
```

```
                bit_5  : 0..1;
                bit_4  : 0..1;
                bit_3  : 0..1;
                bit_2  : 0..1;
                bit_1  : 0..1;
                bit_0  : 0..1;
    end;

    word_def = record                          { word data type }
               CASE int of

                    1 :  ( bits : bit_def); { this double definition    }
                    2 :  ( word : int    ); { allows the user to access }
                                            { the information bit by bit}
               end;                         { or as a word               }

var

    control_word  : int;
    sub_function  : word_def;
    a_reg         : word_def;
    b_reg         : word_def;
    S1            : text;

procedure rte_exec_call $alias 'EXEC'$
                          (request_code : int;
                           var control_request : int;
                           var subfunction  : word_def);  external;


procedure get_the_a_b_registers $alias 'ABREG'$
                          (var a:word_def; var b : word_def); external;


procedure initialize_sub_function;

begin   { initialize optional parameters }

    sub_function.bits.bit_15 := 0;
    sub_function.bits.bit_14 := 0;
    sub_function.bits.bit_13 := 0;
    sub_function.bits.bit_12 := 0;
    sub_function.bits.bit_11 := 0;
    sub_function.bits.bit_10 := 0;
    sub_function.bits.bit_9  := 0;
    sub_function.bits.bit_8  := 0;
    sub_function.bits.bit_7  := 0;
    sub_function.bits.bit_6  := 0;
    sub_function.bits.bit_5  := 0;
    sub_function.bits.bit_4  := 0;
    sub_function.bits.bit_3  := 0;
```

```
    sub_function.bits.bit_2  := 0;
    sub_function.bits.bit_1  := 0;
    sub_function.bits.bit_0  := 0;

 end;

begin    { beginning of main program }

  initialize_sub_function;
  sub_function.bits.bit_1 := 1;
  sub_function.bits.bit_3 := 1;      { set the appropiate bits }
  sub_function.bits.bit_5 := 1;      { in the optional parm    }
  sub_function.bits.bit_7 := 1;
  sub_function.bits.bit_9 := 1;
{
          .
          .
          .
}
{ construct the control word }

  control_word := lu_number+(function_code*shift_left_six_bits);

{ place the RTE EXEC call }

  rte_exec_call (lu_control_request_code,
                       control_word, sub_function);

  get_the_a_b_registers (a_reg,b_reg);


  begin    { examine the A-register for the status of this channel }

       reset (S1,'1');
       {
               .
               .
               .
       }
     if a_reg.bits.bit_15 = 0
     then
        if a_reg.bits.bit_14 = 0
        then
           writeln (S1, 'unit available for use')
        else
           writeln (S1,'unit disabled')
     else
        if a_reg.bits.bit_14 = 0
         then
            writeln (S1,'unit currently in operation')
         else
```

```
            writeln (S1, 'unit waiting for DMA channel');
            {
                .
                .
                .
            }
    end;
end.        { end of PASCAL example }
```

It may be necessary or desirable to use aliases for each EXEC service used in a program for the following reasons:

- The name EXEC represents an entire class of services. A program using EXEC calls will be more readable if a descriptive PASCAL/1000 name is given to each service.
- Each EXEC service requires a different set of parameters. Some services (e.g., EXEC 11) have optional parameters. Each PASCAL/1000 routine must have a specific set of parameters.

EXEC CALL Parameters

Request code ICODE and control word ICNWD, in that order, are the first two parameters of an EXEC call. Other parameters, when needed, are described for each request in the following sections.

Control Requests to the MUX

Control requests have the following general format:

```
CALL EXEC(ICODE,ICNWD,IPARM)
```

Request code ICODE has a value of 3 for all control requests. Each control request has a different function code specified in ICNWD, and the value of the IPARM parameter depends on the function code. Table 2-1 contains a summary of the function codes and their meanings.

Equipment Table word five (EQT5) contains the status word. If a control request is made to an unbuffered EQT, the A-register will contain the device's status. If the EQT is buffered, the A-Register

is meaningless. In either case the B-Register is meaningless, except for control function 6. This status request will return the length of the type-ahead data in the B-register.

When issuing a control function command from the File Manager the following format is used:

```
:CN,lu,fn[,pr]

       lu = LU number
       fn = control function code
       pr = optional control parameter
```

For example, to place LU 19 in type-ahead mode with cancel on break, the following control command is issued.

```
:CN,19,33B,23000B
```

Table 2-1. Control Function Codes

CODE	DESCRIPTION	PARAMETERS REQUIRED
6B	DYNAMIC STATUS OF PORT	NO
20B	ENABLING SCHEDULING	NO
21B	DISABLE SCHEDULING	NO
22B	SET TIMEOUT	YES
23B	BUFFER FLUSH (OUTPUT)	NO
24B	BUFFER UNFLUSH	NO
26B	FLUSH INPUT BUFFER	YES
27B	SET PROGRAM ADDRESS	YES
30B	SET PORT'S ID	YES
33B	CONFIGURE DRIVER RESPONSES	YES
37B	SET READ TYPE	YES

Device Initialization

Device initialization is accomplished by executing the following three control functions to enable terminals. Normally this is done for each terminal from your WELCOM file, but can be executed interactively.

- set port ID (required)
  function 30
- configure driver responses (optional)
  function 33
- enable scheduling (optional)
  function 20

Set Port's ID (Control Function 30)

Control function 30B establishes a logical connection between the logical unit and the physical terminal connected to the interface. This function is normally executed from the WELCOM file to initialize and configure ports 0 through 7, but can be done interactively. Function 30 must be issued before any other request is given to that port. If other commands are sent prior to this function call they may be ignored.

The value of IPARM for control function 30 is defined as follows:

Bits 15-14: # bits per char for transmission and reception. This does not include parity.

0 = 5 bits/char 2 = 6 bits/char
1 = 7 bits/char 3 = 8 bits/char

Bit 13: Reserved for future use and should be set to zero.

Bit 12: Baud rate generator for this port:

0 = baud rate generator 0
1 = baud rate generator 1

Bits 11-10: # stop bits:
All data transfers to and from the interface card require a delay between each character. For all asynchronous transfers there is always at least one stop bit.

0 = reserved 2 = 1-1/2 bits
1 = 1 bit 3 = 2 bits

Bits 9-8: Parity select:

0 = none 2 = none
1 = odd 3 = even

Bit 7: 1 = ENQ/ACK handshake enabled
0 = handshaking disabled

Bits 6-3: BAUD rate:

0 = no change	8 = 1800
1 = 50	9 = 2400
2 = 75	10 = 4800
3 = 110	11 = 9600
4 = 134.5	12 = 19200
5 = 150	13 = reserved
6 = 300	14 = reserved
7 = 1200	15 = reserved

NOTE:

All ports on a Baud Rate Generator must be initialized to the same baud rate. The user should be certain that all ports have their baud rate set using this function, regardless of the baud rate at which the terminal is strapped. In addition, 19200 baud is not supported on 8 channels simultaneously since it would exceed the maximum throughput of the card (78600 baud). A baud rate parameter of zero will not change any of the port's parameters (baud rate, parity, stop bits, etc.).

Bits 2-0: Port number of this terminal (0-7)

For example, to set the port ID of an HP 2621 terminal on channel 0, as LU 41, with a baud rate of 9600, ENQ/ACK handshaking enabled, and no parity the following control request is issued:

```
CN,41,30B,142330B
```

Configure Driver Responses (Control Function 33)

Control function 33B uses the IPARM parameter to specify port parameters. The value of IPARM will configure the driver without sending the parameters to the card. The bit fields are defined as follows (all fields default to the 01 state at system boot time):

Bits 15-14 Reserved for future use, should be set to zero.

Bits 13-12 These bits define the type-ahead feature of the MUX card.

00 = no change

01 = (default value) no type-ahead. Striking any key when there is not a read request pending will gain the system's attention, if enabled.

10 = Type-ahead data can be received without a pending read request. The information on the card is saved until a read request is made. At this point the data is retrieved. Only the break key will gain the system's attention unless type-ahead with scheduling is enabled.

Bits 11-10 These bits define the action to be taken when type-ahead data becomes available. Type-ahead data is defined as "available" when an End-of-Record is read by the card. Valid terminators are defined by the previous read or through control request 37.

00 = no change

01 = (default value) bit 2 is set in the EQT status word.

10 = Bit 2 is set and scheduling is attempted.

Bits 9-8 These bits define the action to be taken when the BREAK key is struck.

00 = no change

01 = (default value) if scheduling has been enabled via control 20, scheduling is attempted.

10 = cancels any type-ahead data and then attempts to schedule the designated program.

Bits 7-6 These bits control the sending of read configuration information to the card. Also see control 37B.

00 = no change

01 = This is a normal read operation. The driver examines bits 10-6 of the control word in the user's EXEC request which specifies a unique read request type to the MUX card.

10 = This will not reconfigure the read operation. When bit 7 is set, the driver overlooks bits 10-6, and only the device driver or a control 37 can modify the read configuration type.

Bits 5-4 Reserved for future use, should be set to zero.

Bits 3-0 These bits define the device driver attached to this port. Device driver number one is the default device driver which passes all of the user's requests directly to the interface driver. Other device drivers are defined at system generation time. The device driver address table $DVTB is established at generation time and defines which device driver has what device driver number associated with it. Exactly one device driver is attached to each port at any time. If zero is entered no change is made.

For example, to configure the driver response for a full type-ahead mode on the terminal/device associated with LU 41: LU 41 has device driver #3 (DDV05) attached to port 0.

```
ICODE = 3
ICNWD = 3300B+41
IPARM = 022400B
CALL EXEC(ICODE,ICNWD,IPARM)
      .
      .
      OR

CN,41,33B,022400B
```

NOTE: If any port is configured for a type-ahead mode, the break key must be struck to schedule the LOGON prompt.

Enable Scheduling (Function 20)

Control function 20B enables the driver to schedule a program on interrupt. The program to be scheduled is specified at generation or the driver can interactively be informed of the address of the program's ID segment through control function 27. Scheduling will commence if the following conditions are met:

1. Scheduling is enabled

2. The program to be scheduled is dormant (state 0)

3. A read operation is not in progress.

4. The port is not in type-ahead mode and any key is hit

   -or-

   The port is in type-ahead mode and the break key is hit (see control 33 regarding type-ahead and the break key).

   -or-

   The port is in the type-ahead mode with "scheduling on data available" and a valid terminator, or count, is received.

To enable scheduling and schedule a program on an unsolicited interrupt the following request is issued:

```
ICODE = 3
ICNWD = 2000B+LU
CALL EXEC(ICODE,ICNWD)
```

or to enable scheduling on an unsolicited interrupt at LU 41:

```
CN,41,20B
```

Other Interface Driver Control Requests

Disable Schedule (Function 21)

Control function 21 resets the flag set by control function 20 (enable scheduling). When a terminal is disabled, striking a key on the keyboard will not schedule the program specified at generation or by control request 27. Once a port is disabled, programs will not be scheduled. At boot-up time the default value of the schedule enable flag is disabled.

For example, to disable LU 41:

```
ICODE = 3
ICNWD = 2100B + 41
CALL EXEC(ICODE,ICNWD)
      .
      .
      OR

:CN,41,21B
```

To re-enable the terminal an "Enable Schedule" request must be issued:

```
:CN,41,20B
```

Set Timeout (Function 22)

To alter the RTE device time-out value that was established at system generation, use control function 22B. The time-out value can be set in 10 millisecond intervals by the integer provided as an additional parameter.

Time-out values can also be set by using the system TO command. The RTE Programmer's Reference Manual describes how to use the TO command. When using a control request or the File Manager CN command, be sure to specify the channel or device by LU number; when you use the TO command, specify the channel by the EQT number. The TO command checks for a lower limit of 500 ms but control function 22 does not.

The timer specifies the number of tens of milliseconds to wait for keyboard input. If this time is exceeded before a user keyboard input completes, the driver sets bit 7 in the terminal's status byte and returns to the caller with a zero length transmission log.

For example, to set the timeout of LU 41 to 25 seconds:

```
ICODE = 3
ICNWD = 2200B + 41
ITO = 100 * 25
REG = EXEC(ICODE,ICNWD,ITO)
      .
      .
      OR

CN,41,22B,2500
```

Buffer Flush (Function 23)

Issuing a control request to a specified LU with a function code of 23B will place that LU's port in the buffer flush condition. In this condition, the driver ignores all write requests for the designated port.

The buffer flush condition is removed automatically when a read request is processed or the pending request queue on the EQT is empty. The buffer flush condition may also be removed by issuing the "Buffer Unflush" control request.

The following is an example of establishing a buffer flush condition on LU 41:

```
      .
      .
ICODE = 3
ICNWD = 2300B + 41
REG = EXEC(ICODE,ICNWD)
      .
      .
      OR

:CN,41,23B
```

Restore Output Processing (Function 24)

To remove a buffer flush condition, issue a control request with a function code of 24B. The Restore Output Processing function restores the ability to process write requests after a buffer flush condition. If the port is not in the buffer flush state, this call is ignored.

The following is an example of restoring output processing to LU 41:

```
      .
      .
      .
ICODE = 3
ICNWD = 2400B + 41
CALL EXEC(ICODE,ICNWD)
      .
      .
      .
      OR

:CN,41,24B
```

Flush Input Buffer (Function 26)

Control function 26B instructs the interface card to clear any data from the channel's input buffer which might have accumulated in type-ahead mode. The value of IPARM indicates whether only the active buffer (IPARM=0) or all of that port's receiving buffers (IPARM=1) should be cleared. Note that this is not the same as Buffer Flush control function 23B. Control 26B ensures the user that the information requested is what is obtained, eliminating the possibility of processing any outstanding data previously entered. To flush only this channel's active buffer use IPARM=0. Setting IPARM=1 will flush the two 254 byte input buffers on this port.

For example, to flush the two input buffers on LU 41:

```
ICODE = 3
ICNWD = 2600B+41
IPARM = 1
CALL EXEC(ICODE,ICNWD,IPARM)
      .
      .
      OR

CN,41,26B,1
```

Set Program Address (Function 27)

Control function 27B saves the value of IPARM as the address of the ID segment of a program to be scheduled on an unsolicited interrupt. If the value of IPARM is zero or negative, program scheduling is disabled regardless of function 20 (octal). This call will override for this

particular port the value set at system generation time. This control function is intended for programmatic rather than interactive use. A program's ID segment address can be found using the system utility subroutine IDGET. Care should be exercised that the address supplied is correct and points to an ID segment of a permanently loaded program. For example, if IADDR contains the address of a program's ID segment, this program will be scheduled on an unsolicited interrupt.

```
ICODE = 3
ICNWD = 2700B+LU
CALL EXEC(ICODE,ICNWD,IADDR)
      .
      .
      OR

CN,LU,27B,IADDR
```

Set Read Type (Function 37)

Control function 37B sends configuration information to the interface card for use in read (EXEC 1) operations. Under normal operation, this information is provided by the interface driver as directed by the function field bits 10-6 of the EXEC request. This call provides the user with a mechanism to override the interface driver defined values or to configure a read operation on the card without executing a read request. This is useful in type-ahead initialization. If bit 7 in the driver configuration word (control 33B) is not set, the next read operation will reset the read type. See Table 2-2 for a description of the SET READ TYPE parameter.

For example, to establish a read type with a buffer transfer on a carriage return for LU 41:

```
ICODE = 3
ICNWD = 3700B+41
ITYPE = 100000B
CALL EXEC(ICODE,ICNWD,ITYPE)
      .
      .
      .
      OR

CN,41,37B,100000B
```

Table 2-2. Set Read Type (Function 37B) Parameter Description.

Bit #	Read Type
15	buffer transfer on a carriage return <CR>
14	buffer transfer on a record separator <RS>
13	buffer transfer on control-D
12	buffer transfer on DC2
10-11	reserved for future use, should be set to zero
9	enables input data editing (backspace and delete)
8	enables input data echoing
0-7	reserved for future use, should be set to zero

Dynamic Status (Function 6)

All requests to the MUX (EXEC 1, 2 and 3 calls) clear bits 3 through 7 of the port's status. Control function 6B can be used to verify if program scheduling is enabled or type-ahead data is available. If program scheduling is enabled bit 1 would be set and if type-ahead data is available, bit 2 would be set in the A-register. Bits 0, 3-7 would all be zero. The B-register can be examined to determine the length of the available type-ahead data.

An example of a dynamic status request to LU 42 is coded as follows:

```
ICODE = 3
ICNWD = 600B+42
CALL EXEC(ICODE,ICNWD)
CALL ABREG(IA,IB)
      .
      .
```

The dynamic status request is a call to the driver; therefore, it will wait until any outstanding requests to that EQT have completed.

Chapter 3
Using the MUX

This chapter deals with using the features of the HP 12792A Multiplexer. Type-ahead is explained, and the most commonly used type-ahead modes are discussed. Error handling and failure analysis in user-written programs are also addressed.

Normal Mode

In the normal, non-type-ahead mode of operation, the subsystem will appear identical to other non-multiplexed RTE terminal drivers. When a port is inactive, the driver will leave an "interrupt on any character" read pending on the card so as to be informed when a key is struck. The appropriate action (system attention, program schedule, etc.) will then be taken.

Type-Ahead

Type-ahead is the ability of a system to accept data from the user's terminal or device before it is requested by the CPU. The MUX card is a buffered device and for each channel it is capable of holding up to two, 254 byte buffers of text in on-board memory.

An advantage of type-ahead is that applications programs can make the system appear more responsive to the user, increasing TOTAL (human included) throughput. This is done by having the application program prompt the user for his/her next response while processing the previous one. By the time the user has finished typing, the system will have processed the last request and can begin on the next. As long as the processing takes less time than the typing, the user perceives instant response time.

While in type-ahead mode, the driver leaves a read request pending on the CARD (not the EQT) at all times. This read allows the user to enter data into the card even though the SYSTEM does not have a read pending. Upon receiving a record the card will interrupt the CPU, telling it that a buffer of data is available. If no request has been

posted to that port, a flag is set in the status word and the driver returns to the system and waits. When a request is issued, the driver reads the data from the card and completes the user request.

Since keyboard characters are buffered on the card, system attention in type-ahead mode cannot be gained by striking a terminal key. The BREAK key, however, is not buffered and can be used to obtain system attention.

Since multi-line type-ahead is possible, two different type-ahead modes are available. Full type-ahead, as described above, would cause successive read requests to fetch successive lines of text from the multiplexer card. This mode is useful for such tasks as text editing and using DBUGR. Typing can be done as far ahead of the data processing as allowed by available multiplexer buffer memory, up to and not exceeding two records.

In situations where system response could radically alter a user's next command (FMGR error messages, for example) a full multi-line type-ahead may cause problems. The following will illustrate this problem:

User types... ST,FILE,8

while tape is moving, the user types:

PU,FILE

the tape runs out; the system downs the device

The user hits the BREAK key, the system issues a prompt and a read, the system rather than FMGR reads the PU command from card buffer, and tries to execute the FMGR command.

In the above example, the user merely gets back an "OP CODE ERR" from the system the first time the request for system attention is made. It is possible, however, for the commands stored on the card to have a disastrous effect on the system.

The solution to the above problem is to configure the driver to cancel all card data upon receiving a BREAK interrupt (see control function 33B). This preserves the multi-line type-ahead feature, and reduces the chance of data being read by the wrong process.

If an analogous situation could occur for user written programs, another possible solution is for the user to issue a Flush Card Buffer request (control 26B,1) prior to any sensitive read request. This will clear the extra commands before they can be misread.

Note that type-ahead is also useful in non-terminal device communication. The buffering on the card eliminates the need for

stacking two or three class read requests on an LU to prevent data loss, thus reducing program size and complexity and the need for lots of SAM. Type-ahead scheduling can be used to invoke a data processing program.

When data is available on the multiplexer card and there is no pending request to accept it, a bit is set in the status word and program scheduling will be attempted. Should the user program decide it doesn't want the data, it can issue an input flush (control 26B) to remove the data.

Program Scheduling

Program scheduling is a mechanism whereby certain external events will cause the interface driver to schedule a program in the system. This program is given the address of EQT word 4 which provides adequate information to determine at which port the event occurred.

Program scheduling is an optional feature which must be enabled to operate. A set of control requests (Control 20B, 21B) are used to enable and disable this feature. An additional request (Control 27B) is provided to allow programmatic setting of which program is to be scheduled, overriding the program designated when the system was generated.

There are several events which can cause a program to be scheduled. If the port has been set to "normal" mode (see Control 33B, bits 13, 12) striking any character key or the BREAK key on a terminal keyboard (or the receipt of a character or BREAK from a device) will cause the character to be ignored and the designated program to be scheduled.

A port which has been configured in type-ahead mode will require the receipt of a BREAK to schedule the program, because any characters received will be saved in the port's data buffers. The port can also be configured to schedule the program on receipt of a buffer of data from the device. This type-ahead with scheduling mode will still recognize the BREAK key.

A program scheduled by one of the above events is run with the machine's B-register pointing to word 4 of the Equipment Table (EQT) entry corresponding to the port upon which the event took place. Library functions EQLU or TRMLU can be used by the program to recover the Logical Unit (LU) of that port. This LU can be used later to make I/O, Control, and status calls to that port. The library subroutine RMPAR can be used to recover EQT words 4 through 8. Note that both these routines require that the B-register set by the driver remain intact.

The programs PRMPT and RPN are used for terminal break-mode command processing, and are supplied with most RTE systems. These programs are an example of the use of program scheduling. In operation, PRMPT is the program which gets scheduled on interrupt from the terminal. When invoked it finds the LU of the interrupting port and writes the break-mode prompt to that LU. It then posts a class read against that port and schedules the program RPN. RPN, waiting on a class GET, receives a command from the user and executes it, returning to suspend on the GET call for the next command.

User supplied programs may be written to process other schedule-driven applications. These programs may either be included as the program to be scheduled when the system is generated, or assigned on-line by a control request.

Common Type-Ahead Modes

Control Function 33 allows the user to control the driver responses by manipulating four fields, specifying sixteen different type-ahead combinations. However, the four following combinations are sufficient for most user applications. If a specific application is desired, the user can design a mode by manipulating the control word's bit fields using control function 33.

The common operating modes are:

Normal Non Type-Ahead Mode

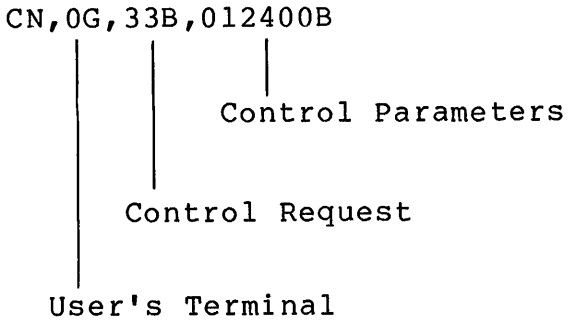
Full Type-Ahead Mode

Type-Ahead with Flush on Break Mode

Type-Ahead with Scheduling Mode

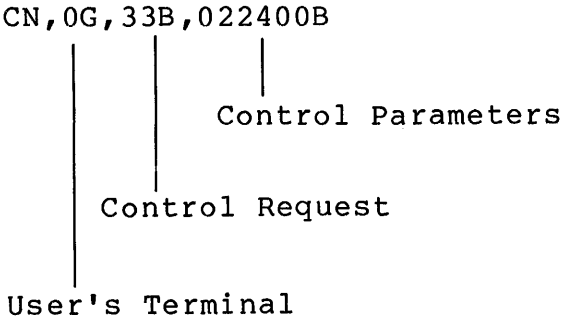
Each mode is summarized below along with the run string necessary to configure the user's terminal. Control requests and bit manipulation are discussed in the Interface Driver section.

No Type-Ahead Mode



This will return the port to a standard RTE mode. This mode is commonly used in the WELCOM file so that the individual may later configure the port to a specific application. If a key is struck while executing in this mode and no request is pending on the terminal, the designated program will be scheduled.

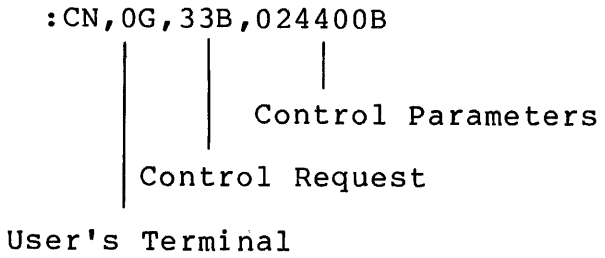
Full Type-Ahead Mode



As discussed earlier, type-ahead is a mode of operation enabling the interface to accept strings of data from terminal devices even though the system did not request any information. This information is stored in one of the two 254 byte receiving buffers for that port and remains buffered until a program reads the contents of the buffer. While in type-ahead the driver leaves a read pending on the interface, looking for a valid terminator. The terminator merely signals the interface driver that a complete record has been encountered, and the interface card will interrupt the CPU.

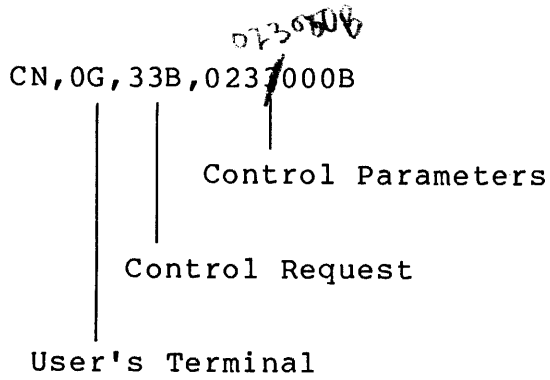
In this mode, the only way to schedule the designated program is to strike the break key. Data resident on the card, if any, is not disturbed.

Type-Ahead with Scheduling Mode



This mode configures the interface driver responses to utilize the type-ahead mode. Scheduling will occur when an end-of-record is encountered. Depressing the Break Key will always schedule the designated program. Program specification is accomplished with control function 27 or, more commonly, it is specified at generation time.

Type-Ahead with Flush on Break Mode



In this mode when the break key is struck, the interface driver will flush the contents of the input buffers and then the program designated by control function 27 is scheduled. This mode of operation is the preferred type-ahead mode because it reduces the possibility of having data misread. The user has the option of "erasing" the contents of the buffers just entered by hitting the break key or leaving the buffers alone allowing them to execute when the next read request reads them.

NOTE: Flush on break may be used in conjunction with either full type-ahead or type-ahead with scheduling.

Error Recovery

Dynamic status checking and I/O status checking allow the user to check on the status of a Multiplexer port for normal processing and error checking. I/O status can provide the user with two of the device status words (EQT5 and EQT4) and an LU status word. Dynamic status checking provides the user with the port's status and the length of any type-ahead data if present. Control function 6 is described in the User Interface chapter.

All errors associated with the Mux will appear as:

- time outs
- parity error or overflow

Parity and overflow errors are indistinguishable. As soon as an error is encountered, the user's buffer is flushed.

I/O Status

The I/O status request, using a request code (ICODE) of 13, calls the RTE operating system to provide information contained in system tables. This EXEC call is not a call to the interface driver therefore bits 3-7 of ISTAT1 are not cleared. The LU number of the port must be specified in the control word (ICNWD). One additional parameter is required and two more are optional. One, two, or three words are returned to the user's program in the parameters passed. Table 3-1 lists the I/O status request returns.

A sample calling sequence for LU 41 is shown below:

```
      .
      .
      .
ICODE = 13
ICNWD = 41
CALL EXEC(ICODE,ICNWD,ISTA1,ISTA2,ISTA3)
      .
      .
      .
```

When the call completes, the variables ISTA1, ISTA2, and ISTA3 contain the I/O status as shown in Table 3-1. ISTA1 is the status word (EQT5) and ISTA2 is also a status word (EQT4).

Table 3-1. I/O Status-Request Returns

Word	Bits	Description
ISTA1	15-14	I/O controller availability indicator: 00 = Available for use 01 = EQT disabled (down) 10 = Device busy
	13-8	Equipment Type code
	7-0	Status: 1 =
	7	Last request timed out
	6	Break Key hit
	5	Control D entered on the last request (EOT)
	4	Reserved for future use, always zero
	3	Parity error or overflow detected on the last request
	2	Type-ahead data available, control function 6B will return the length of type-ahead data in the B-Register
	1	Program schedule enabled
	0	Reserved for future use, always zero
ISTA2	15	Reserved for future use, always zero
	14	1 = Automatic output buffering enabled
	13	1 = Driver to process power fail (always 1)
	12	1 = Driver to process time-out (always 1)
	11	System communication flag
	10-6	Last subchannel addressed
	5-0	Select code of the Multiplexer Card
ISTA3		Logical Unit Status:
	15	1 = LU down, 0 = LU up
	14-5	Reserved for future use, should always be zero
	4-0	EQT subchannel associated with the LU number

Failure Analysis

For failure analysis it is important to note that all errors appear as time outs, parity or overflow errors. Current HP supplied device drivers will not produce the following error message. User written device drivers may use the EXIT command described in Chapter 4 to have the following error message displayed at the system console.

```
I/O XX L #x E #Y S #z

XX = NR
     TO
     PE
     etc., as specified by device driver
```

In the above example x, y, and z are the Logical Unit number, EQT number, and subchannel number respectively. If a device is down, any I/O control request will wait for the operator to "UP" the EQT number.

For time-outs it may be that a simple time-out has occurred, indicating the operator was too slow in responding to a read request. In this case, retry the request.

If the device is not downed, control will return to the user program on error and the user program should be structured to check for errors and process accordingly. The "I/O Status" request can be used to obtain the status to test for various conditions.

Read Errors

If the last command issued was a read, a parity error or data overrun may have occurred. Data Overrun indicates the Multiplexer maximum throughput rate has been exceeded and data on the LU with the overflow bit set was lost. These errors are only detectable on a read operation.

Chapter 4
Device Driver Writing

This chapter explains the writing and use of device drivers that can be called by the 12792A Multiplexer interface driver, DVM00. The basic philosophy of using device drivers is explained to give the user a better understanding of the steps involved.

Device Driver/Interface Driver Concept

An interface driver is a standard RTE driver that converts user EXEC requests for input, output, and control into a sequence of assembly level instructions which control and pass data to an interface card through the I/O backplane of an HP-1000 computer. The interface driver need know nothing about the device that is the eventual source or destination for the data; the interface driver only communicates with the interface. The device driver modifies user requests to make them compatible with the device. The device driver is a subroutine that is called by the interface driver to examine and modify user requests.

Reasons for Device Driver/Interface Driver Use

The device driver/interface driver concept offers several advantages over the conventional monolithic driver. The use of device drivers allows flexibility for system designers and users. Many different types of equipment may be controlled by a single type of interface as long as they are electrically compatible and use the same basic line protocol. For example, any RS-232-C (electrical specification), asynchronous (basic line protocol) device may be controlled by the 12792A interface. Individual device drivers for each of these devices may be easily written without knowledge of the I/O card/backplane interface. New devices may be added to a system without undertaking the monumental task of writing an entirely new driver. The system programmer need only write a subroutine to add the required character sequences to the user data to control the new device.

Differences between devices may be made transparent to user programs. Applications programmers need only concern themselves with reading or writing data to a "standard" device while the device driver takes care of the control needed for the "exotic" device the program is actually communicating with.

Some devices may be customized by using different device drivers for different tasks. Printing terminals may be made to look like line printers to the user program by writing a device driver to translate column one carriage control into the proper escape sequences for the terminal. A different device driver may then be used when an interactive terminal is desired. These various device drivers may be dynamically switched in and out by the user program or by the system manager when required.

A single driver written to control a large number of different devices through a common type of interface would be very large. Requiring the use of this driver would penalize users who only need a few of these devices. By using device drivers a system manager need only include the driver code needed for the devices on the system, thus saving space for other uses.

Interface Tasks

To write efficient device drivers it helps to have an understanding of the responsibilities of the other components in the I/O interfacing subsystem. In the HP 12792A Multiplexer Subsystem the HP 12792A interface card is primarily responsible for sending and receiving characters on the RS-232-C line and for handling line protocol. When enabled to do so, the interface card will handle the ENQ/ACK line protocol to prevent terminal buffer over-runs. The interface card will transmit and receive the characters to and from the terminal at the baud rate for which it has been set by the driver. The interface card will automatically pack the eight bit characters into sixteen bit data words for efficient DMA transfer to the computer, or unpack the sixteen bit words into eight bit bytes for the terminal. The interface card will treat the parity bit as described in the Interface Support statement in the Multiplexer Configuration Guide, and will notify the driver when incorrect parity has been received.

Interface Driver Tasks

Interface Control

The interface driver (DVM00) is responsible for controlling the interface card via assembly level I/O instructions to the computer I/O backplane. The interface driver interprets user requests to properly initialize the interface card for baud rate, parity, character length, number of stop bits, etc. The interface driver initializes and starts DMA transfers between the computer memory and the interface card.

Operating System Interface

The interface driver receives EXEC level user requests from the RTE operating system and passes them to the device driver for further processing. The interface driver processes requests from the device driver, returning to the device driver on each request completion. The interface driver requests a DMA channel from the RTE operating system when a data transfer is required either to send data to the card or receive data from the card. When the device driver informs the interface driver that the user request is complete, the interface driver returns to RTE with the correct device status and transmission log or error code in the A and B registers.

Device Driver Tasks

The device driver is entered on each new user request and on completion of each device driver request. The device driver may do further checking on request legality. If the device requires a special sequence of characters prior to receiving or sending the user data, the device driver should format the characters into a buffer and send them to the device via a device driver request to the interface driver. When the user request is to be processed the device driver tells the interface driver to start the request currently in the EQT. When the entire request has completed, the device driver places the correct status in the EQT and the transmission log in the B register, and then informs the interface driver that the request is complete.

HP Implementations of Device Drivers

There are two examples of device driver applications that Hewlett-Packard has implemented as part of the HP 12792A Multiplexer subsystem.

Lineprinter Device Driver DDV12

The device driver DDV12 is used to make HP 2631/2635/7310 printers look like typical line printers to user programs. These devices use escape sequences and control characters for carriage control while standard line printers interpret the first character of each line as a carriage control character. The DDV12 device driver examines the user's first character and sends the proper control character sequence to the printer. The first character is then stripped from the data and the data is sent to the printer. The device driver also changes the driver type in the EQT to type 12 for lineprinters.

Block Mode Terminal Device Driver DDV05

The device driver DDV05 allows utilization of the block mode read capabilities of an HP 264X or 262X terminal. The first time a read request is made to the terminal its status is read to determine whether it is in block or character mode. When a user read request is made the device driver first issues a write to the interface driver to send a DC1 to the terminal. If the terminal is in block mode the device driver issues a read to the interface driver and examines the first returned character. If the character is a DC2 the device driver knows that the terminal is in block mode and trying to send data to the interface. A read is then issued for the user buffer length with echo and character editing turned off. If the returned character is not a DC2, the user program had probably issued a program enabled block read (escape lower case d) and the program's data is in the buffer just read.

If the terminal is in character mode, the user's request is executed as-is after the DC1 is sent (to enable the softkeys).

Device Driver Interface

An uncomplicated device driver/interface driver interface is provided making it easy for systems programmers to write their own device drivers. All that is required beyond the information given in this chapter is a basic familiarity with the flow of I/O requests in RTE and a thorough knowledge of the particular device that is being communicated with.

Device Drivers for HP 12792A Multiplexer

The following points should be kept in mind when writing device drivers for the HP 12792A Multiplexer subsystem. First, all read, write, and control requests are passed to the device driver by the interface driver for modification before they are sent to the interface card. The device driver only has to make read, write, or control requests to the interface driver; the device driver does not issue I/O instructions to the interface card. The device driver requests are at the EXEC level, that is, a request word (control word as defined for EQT word 6) buffer address and length, or optional control request parameters are passed to the interface driver. A device driver will typically make several of these requests for each user EXEC request. After each device driver request completes the interface driver will return to the device driver for the next request. It is up to the device driver to tell the interface driver when the original user request is complete. The device driver and interface driver pass parameters back and forth between each other using the A and B registers and the portion of the EQT extent defined as the Device Driver EQT Extent.

Restrictions and Requirements

When a device driver issues read or write requests the data buffer must be either within the device driver or in the same map as the user request buffer. If the user request is unbuffered the user request buffer is in the user map. If the user request is buffered, class I/O, REIO (re-entrant I/O), or $XSIO (system I/O) the user request buffer is in the system map.

All requests are passed to the device driver prior to checking by the interface driver. Device drivers should check only for control requests that are defined for the device driver. Any unrecognized control requests should be passed on to the interface driver with a command to execute the request.

System Abort Requests

A system abort request may be issued to the interface driver at any time if the user program currently doing I/O is aborted for any reason. The device driver may be entered with the system abort either as a new user request or as a continuation request. If the device driver is entered as a new user request with the system abort request, the device driver should treat it as any unknown control request and pass it back to the interface driver.

However, if the device driver is entered as a continuation with the system abort two problems may occur. If the device driver does not check for the system abort on a continuation entry and subsequently issues a read or write request to the user buffer, the buffer area may have been re-allocated for other uses. Program corruption and system or subsystem crashes may occur. It is also possible to leave the device in an unknown state if an expected user buffer is not written to the device. If this is a problem with a device, the device driver should check for system abort requests and reset the device to a known state.

In all cases the device driver should check for a system abort request prior to issuing an I/O request to the user buffer area.

To test for a device driver system abort request check the contents of EQT word 6 or word 2 of the extent for a 100003 octal ($XSIO control zero request). The above tests are necessary if the device driver issues it's own read or write to or from the user buffer area on a continuation entry. If the device driver is simply trying to execute the user's original request by leaving it in the device driver EQT extent word 2, the test for system abort request is not necessary. In this case the contents of device driver EQT extent word 2 are changed to a control zero at the same time as EQT word 6.

Interface Definitions

Entry to the Device Driver

On entry to the device driver the following parameter locations are defined:

A register, Bit 15: 1 = initial entry on a new user request
0 = continuation entry, signifying a previous device driver request is complete

Bits 14-0: address of device driver EQT extent (defined below)

B register: Previous device driver request transmission log, if any

The device driver EQT extent words 2-4 are set to the current user request definition. These three words are copied from EQT words 6-8 on each (new or continuation) entry to the device driver. See the expanded definitions of the EQT words below.

Base page locations 1660 through 1672 are the addresses of the current EQT words 1 though 11 and base page locations 1771 through 1774 are the addresses of EQT words 12 through 15.

On each entry EQT words 4-10 and 14 are defined per the RTE definitions (expanded below). EQT words 9 and 10 (READ/WRITE optional parameters) are defined per RTE on new request entries only (A register bit 15 = 1). However, they are not defined on the subsequent continuation entries. If their contents are required by the device driver on subsequent entries, they should be saved in the device driver EQT extent on the new request entry.

Return to the Interface Driver

On return to the interface driver the device driver must insure that the proper parameters are passed back to the interface driver. The device driver must differentiate between new user request entries and continuation entries as the parameters returned to the interface driver are different for each case. Also, the device driver must tell the interface driver when the original user request is complete and set up the correct transmission log and status indications for the calling program.

On return to the interface driver the following parameter locations are defined:

A register, Bits 15-3: Function modifier

Bits 2-0: Exit command

B register: Request timer, or user transmission log

EQT 5, Bits 7-0: Status for user

Device driver EQT extent,

word 1: Physical record length in characters for read requests if different from user buffer length

word 2: Request word as defined for EQT word 6 except that bits 15-11 are not defined and should be zero

word 3: Request buffer address

word 4: Request buffer length (positive number of words or negative number of characters)

Return to Interface Driver — Device Driver EQT Extent

The physical record length (device driver EQT extent word 1) is used to prepare the 12792A interface card for binary data read requests where the device does not terminate the record with a special character such as carriage return. The physical record length must be a positive number of characters. If this parameter is not set it defaults to the user buffer length.

Words 2-4 of the device driver EQT extent may be changed by the device driver to cause the interface driver to execute some other request, or they may remain unmodified causing the interface driver to perform the initial user request. Remember that on each entry words 2-4 of the device driver EQT extent are restored to the original user request copied from EQT words 6-8.

Return to Interface Driver — A-Register

Exit Command

On return to the interface driver from the device driver the A register bits 2-0 must be set up with the exit command. The exit command definition is similar to the RTE definition for the A register for a standard driver on return to RTE. The exit command definition is as follows:

```
Exit command if entered with a new user request
(A register bit 15 = 1 on entry)

  0 = start request in device driver extent words 2-4;
      B register = time out value (-10's ms)
  1 = user I/O request is illegal, give IO07 error
  2 = user control request is illegal, ignore it
  3 = I/O device not ready, down it and print IONR message
  4 = user request completed (immediate completion);
      B register = transmission log
  5 = start request (same as 0)

Exit command if entered after completion of a device driver
request (A register bit 15 = 0 on entry)

  0 = user request is complete; B register = transmission log
  1 = I/O device not ready, down it and give IONR message
  2 = end of transmission (EOT) reached, down device and
      give IOET message
  3 = parity error, down device and give IOPE message
  4 = device time out, down device and give IOTO message
  5 = new request in device driver EQT extent words 2-4;
      B register = time out value (-10's ms)
```

Function Modifier

Error type exit commands (new request 1, 2, and 3 or continuation entry 1, 2, 3, and 4) are simply passed to RTE in the A register by the interface driver. Action taken (program abort, print error message, etc.) is determined by RTE the same as for any standard driver.

For exit commands that initiate another device driver request (new request exit command = 0 or 5, continuation entry exit command = 5) a function modifier may be placed in the A register bits 15-3 to override and expand the normal request function code contained in the device driver EQT extent word 2, bits 10-6. Function modifiers are defined for read and write operations. Write function modifiers contain read modification fields and can be used to define the function modification for the next read or series of read operations. The write function modifier bit fields are defined as follows:

A register bits 15-3

Bit 15: end transfer on carriage return (CR)
14: end transfer on record separator (RS)
13: end transfer on end of tape (EOT, control D)
12: end transfer on (DC2)
11: end transfer on specified character count
10: enable end on character specified in bits 15-12
9: enable character editing (backspace, delete, etc)
8: echo received characters
7: not defined, should be 0
6: not defined, should be 0
5: disable ENQ/ACK handshake this transfer only
4: add CR/LF to buffer if last character is not an underline (137 octal)
3: use write overrides in bits 7-4, if bit 3 is 0, bits 7-4 should be 0

If bits 7-3 are zero (do not override) the write is configured by bits 10-6 in the device driver EQT extent word 2. These bits are defined as bits 10-6 in EQT word 6 and the ICNWD description in the EXEC write section of this manual.

The read related fields in the write function modifier (bits 15-8) will configure the card for any subsequent read operations. This eliminates the "window" between writing and reading so that if the write triggers a response from the device no data will be lost.

If enable end on character (bit 10) is set, one or more of bits 15-12 must be set. Reads will complete on reception of any one of the specified characters. If bit 10 is clear, bits 15-12 should be zero. If end on count (bit 11) is set, the physical record length in device driver EQT extent word 1 should be set to a positive number of characters. If bit 11 is set, the end on character bits 15-12 and 10 are ignored. Only one of bits 10,11 should be set.

For read functions the modifier bit fields are defined as follows:

A register bits 15-8,3

Bit 15: end transfer on carriage return (CR)
14: end transfer on record separator (RS)
13: end transfer on end of tape (EOT, control D)
12: end transfer on (DC2)
11: end transfer on specified character count
10: enable end on character specified in bits 15-12
9: enable character editing (backspace, delete, etc)
8: echo received characters
7-4: reserved for future use, should be zero
3: use current card configuration

If enable end on character (bit 10) is set, one or more of bits 15-12 must be set. Reads will complete on reception of any one of the specified characters. If bit 10 is clear, bits 15-12 should be zero. If end on count (bit 11) is set, the physical record length in device driver EQT extent word 1 should be set to a positive number of characters.

If bits 15-3 are all zero the read configuration will be determined from the control word bits (10-6) in the device driver EQT extent word 2. These bits are defined as in EQT word 6 and the CONWD described in the EXEC read section of this manual. Bit 9 of the EXEC request (EQT 6) is always used and must be valid. If bit 3 is set bits 15-4 should be zero.

Return to Interface Driver — B-Register

On return to the interface driver the device driver should set the B register to be either the transmission log, or the device time out value. If the initial user request is complete, the B register should contain the transmission log to be returned to the user program. Transmission logs for each device driver operation are returned to the device driver in the B register by the interface driver on each continuation entry. The device driver may save one or more of these to return, or the device driver may return any meaningful number. Convention requires that the transmission log be a positive number, either a number of words if the initial user request specified words (EQT word 8 is positive) or a number of characters if characters were initially specified (EQT word 8 is negative).

If a new device driver request is to be initiated (exit command = 5 on continuation, or 5 or 0 on initial entry) the B register should contain the request time out value. This value will be a negative number of time base ticks (10's of milliseconds). If the device driver needs to use the system defined time out for the device, EQT word 14 should be copied into the B register.

Return to Interface Driver — EQT Entries

As a general rule it is not advisable for the device driver to modify the EQT, except for the area defined as the device driver EQT extent. However some EQT areas are routinely modified by device drivers. In EQT word 5 the equipment type (bits 13-8) should be modified on the first entry to a device driver to reflect the device type the device driver is emulating. On each completion exit (new request exit command = 4, or continuation entry exit command = 0) the status field in EQT word 5 bits 7-0 should be updated to return status to the user. This field should be the same as is defined for the device type the device driver is emulating.

After the user's request has been processed, if further interaction with the interface driver is required EQT words 7 and 8 are available as convenient temporary storage areas. It is common to store the transmission log in EQT word 8 in these cases. Words 9 and 10 are used by the interface driver for temporary storage and should not be modified by the device driver.

Selected EQT Definitions and Uses

EQT Word 4 -- Subchannel

EQT word 4 bits 10-6 contain the currently addressed subchannel. This information is required by device drivers that perform different tasks for different subchannels.

EQT Word 5 -- Equipment Type Code and Status

EQT word 5 bits 13-8 contain the equipment type code as specified by the driver name at generation. The 12792A Multiplexer interface driver is DVM00 so the type code is 00. Device drivers that emulate devices should use a type code that corresponds to the device they are emulating. On first entry the device driver should change the type code in the EQT table. The following is a list of type codes and devices they represent:

00 to 07 = terminals or paper tape devices
- 00 = teleprinter or keyboard control device
- 01 = photoreader
- 02 = paper tape punch
- 05 = intelligent terminal devices generally having block mode capability (HP 264x and 262x terminals)
- 07 = multipoint devices

10 to 17 = other unit record devices
- 10 = plotters (Calcomp or HP 7210)
- 11 = card readers
- 12 = line printers
- 13 = TV monitor
- 15 = mark sense card readers

20 to 35 = mag tape or mass storage devices
- 23 = 9 track mag tape
- 24 = 7 track mag tape
- 30 = fixed head disc
- 31 = 7900 moving head disc
- 32 = 7905/6/20/25 moving head disc
- 33 = flexible disc drives
- 36 = writable control store (microcode execution space)
- 37 = HPIB Interface
- 47 = Multidrop FDL interface

EQT word 5 bits 7-0 contain the device status of the 12792A Multiplexer on each entry to the device driver. This status is defined as follows:

Bit 7 Time out, if the driver was entered on a time out this bit will be set on entry to the device driver.

6 Break, if the 12792A Multiplexer card received a break character from the terminal during the last operation this bit will be set.

5 EOT, The End Of Tape bit will be set if an EOT (004 octal) was received during the last read.

4 Bit 4 is not currently used

3 PE/OV, the Parity Error/Overflow bit will be set if either one of these conditions occurred on the last read.

2 Type-ahead data, this bit will be set if type-ahead data is available on the card.

1 Schedule, this bit indicates that program scheduling on unsolicited interrupt has been enabled (interface driver control function 20)

0 Bit 0 is not currently used

The above definitions apply whenever the device driver is entered from the interface driver. The device driver is free to change any of the status bits if emulation of other driver types is desired. On a user request complete exit from the device driver the status bits (EQT word 5) will be passed to the user program in the A register.

EQT word 6 and device driver EQT extent word 2 contain the current request word which is defined as follows:

Bits 15-14 Request type:
00 = standard user request
01 = automatic buffered user request (request buffer is in system available memory)
10 = a system request ($XSIO)
11 = user Class I/O request (request buffer is in system available memory)

Bits 15-14 are only defined in EQT word 6. They are undefined in the device driver EQT extent word 2 and should be set to zero if the device driver modifies this word.

Bit 12 Z-bit indicates a second buffer is available on a read or write. If set, EQT word 9 contains the address of the buffer and EQT word 10 contains the length. If the Z-bit is clear, EQT words 9 and 10 contain 1 word optional parameters.

In the 12792A Multiplexer subsystem, the interface driver does not use the double buffering feature, it is therefore available to the device driver for use.

Bits 10-6 Subfunction, as defined in the EXEC request section of this manual for the EXEC control word.

Bits 1-0 Function:
01 = read request
10 = write request
11 = control request

Bits 13,11 and 5-2 These bits are undefined and should be set to zero if the device driver modifies the request in the device driver EQT extent word 2.

EQT word 7 and device driver EQT extent word 3 is the user buffer address. The interface/device driver is always entered in the same map as the user buffer, so the user buffer address is in the current map.

EQT word 8 and device driver EQT extent word 4 is the user buffer length. The length is either a positive number of words, or a negative number of characters.

EQT word 9 is an optional parameter. If the Z-bit (EQT word 6 bit 12) is set EQT word 9 is the address of a secondary user buffer which is in the same map as the primary user buffer. If the Z-bit is clear EQT word 9 contains the optional parameter or zero if no parameter was passed.

EQT word 10 is an optional parameter. If the Z-bit is set EQT word 10 is the length of the secondary user buffer. It is a positive number of words, or a negative number of characters. If the Z-bit is clear, EQT word 10 contains the second optional parameter, or zero if a second optional parameter was not passed.

Both EQT words 9 and 10 are only available to the device driver on a new request entry. These words must be saved in the device driver EQT extent if they are required later by the device driver.

EQT word 14 contains the system defined time-out reset value for the device, a negative number of time base ticks (10's of ms). This value is set by the system at generation, or by the system TO command, or by the interface driver by a control 22 request. This word may also be set directly by the device driver if desired.

EQT word 15 contains the time out clock count down counter. This word is setup by the interface driver prior to returning to the system. This word should not be modified by the device driver. On any device driver request to the interface driver the time out count for EQT 15 should be passed in the B register. The value should be a negative number of 10's of milliseconds. If the system defined time out is to be used, the device driver must pass the contents of EQT word 14 to the interface driver in the B register.

Device Driver Address Table

The interface driver uses a device driver address table to find the correct device driver when the device driver is selected with a control 33 request. The device drivers are selected by numbers which are determined by their positions in the device driver address table. Each device driver to be used with the interface driver must have an entry in the device driver address table. To add device drivers to the device driver address table, the user must create his own table.

The device driver address table should have the following format:

```
        NAM $DVTB,8    DEVICE DRIVER ADDRESS TABLE
        ENT $DVTB
        EXT DVNM1,...,DVNMn
*
*    DEVICE DRIVER ADDRESS TABLE
*
$DVTB   DEC n          NUMBER OF ENTRIES IN TABLE
        DEF DVNM1+0    ADDRESS OF DEVICE DRIVER   2
        DEF..... +0      . . . . . .              3
    .
    .
        DEF DVNMn+0    ADDRESS OF DEVICE DRIVER   n+1
        END
```

The names of the device drivers may be any valid label, as long as they do not conflict with any other symbol in the system. Note that the first device driver in the table is selected by a control 33 request to use device driver number two. This is because the the value zero is reserved for "no change", and one is the used for the default device driver. Since the device driver number field is 4 bits wide, the user is able to include the default and up to 14 other device drivers in the system.

Location and Size of Device Drivers

Since the device driver address table and device drivers themselves are called directly by the interface driver, they must be resident within the same map. This poses a few restrictions on the number and location of these modules.

The interface driver requires approximately 1400 words of memory, so up to 600 words are left in a standard two page driver partition for the device driver address table and the device drivers. If this is not enough room either the driver partition can be changed to three or more pages, or one or more device drivers and the table may be relocated into Table Area I. If $DVTB is relocated into Table Area I, all device drivers will be forced to Table Area I. The disadvantages are that the user available space in the system is reduced. If the driver partition size is increased, the size of the largest available user partition is reduced by an equal amount, and the size change must be an incremental number of pages. If the modules are relocated in Table Area I, the actual space used may not take away from user space unless a page boundary is crossed, in which case a page will be taken away from the largest available user partition. Device drivers relocated in Table Area I will take space otherwise used as System Available Memory. In the RTE-IVB operating system, only these two

methods of gaining space for device drivers will guarantee that the device driver and interface driver will be in the same map and be mapped properly to handle all user requests. The interface driver and device drivers and table could be all relocated into the system driver area (SDA). More space is available to users, but large background programs will not be able to make unbuffered requests to the driver.

In RTE-M the device drivers and device driver table are simply relocated as system modules along with the interface driver.

Case Study: A Device Driver Writing Example

The following is an example of device driver writing that illustrates some of the problems, solutions, and steps involved in writing a typical device driver. The device driver in this example is written to make a terminal look like two separate terminals, sharing the keyboard, and splitting the screen into two separate areas.

Task Definition

The tasks involved in interfacing with a device using a device driver should be clearly defined and broken down into a sequence of logical steps. In this example the object is to make an HP 26XX terminal appear as two terminals for read and write requests. Requests made to an LU defined as an EQT subchannel 0 will go to the left half of the terminal display, and requests to subchannel 1 will go to the right half of the display.

Three major tasks are defined for the device driver while the interface driver handles the user's actual read or write request. The three tasks all involve sending specific character sequences to the terminal for initialization in a classic device driver application.

Margin Set Up

The first task is to set the left and right margins on the terminal to keep the following text on the respective side of the screen. Upon determination of the subchannel for each new user request the device driver sends the escape sequence to set the left and right margin at predetermined columns on the terminal screen. Due to terminal idiosyncrasies the left margin must be set first for subchannel 0 (left side) and the right margin must be set first for subchannel 1 (right side).

Cursor Position

The second major task is to position the cursor so that the subsequent read or write operations will appear at the correct place on the screen. The escape sequence to position the cursor is formatted with the correct cursor position for the left or right side of the screen. The device driver keeps track of the current cursor position for each side in the device driver EQT extent. Once the request buffer is formatted with the correct cursor position the device driver passes it to the interface driver as a device driver request.

Cursor Tracking

The third major task for the device driver is to find out where the user's request has left the cursor for the side of the screen that was just addressed. To do this the device driver writes a request for a cursor position sense to the terminal and then reads back the result. The resulting cursor position is separated from the escape sequence that precedes it and stored away in the device driver EQT extent for the subchannel that is addressed.

Minor Tasks

Additionally, in an effort to reduce overhead, the device driver is written to not set the margins or position the cursor when the subchannel addressed is the same as the previous subchannel. It is assumed that the cursor and margins will remain in position between sequential requests to the subchannel. In order to implement this step the device driver saves the last addressed subchannel in the device driver EQT extent.

Finally the device driver patches the equipment type code into EQT word 5. This patch will take place only the first time the device driver is accessed for any particular EQT. Since either subchannel most closely resembled type 00 devices to the user the equipment type used is 00. This step is included for illustration only. When any "Select New Device Driver" request is made (CN,LU,30B,XXXXXn where bits 3-0 = 0) the driver is reset to 00.

Device Driver Operation

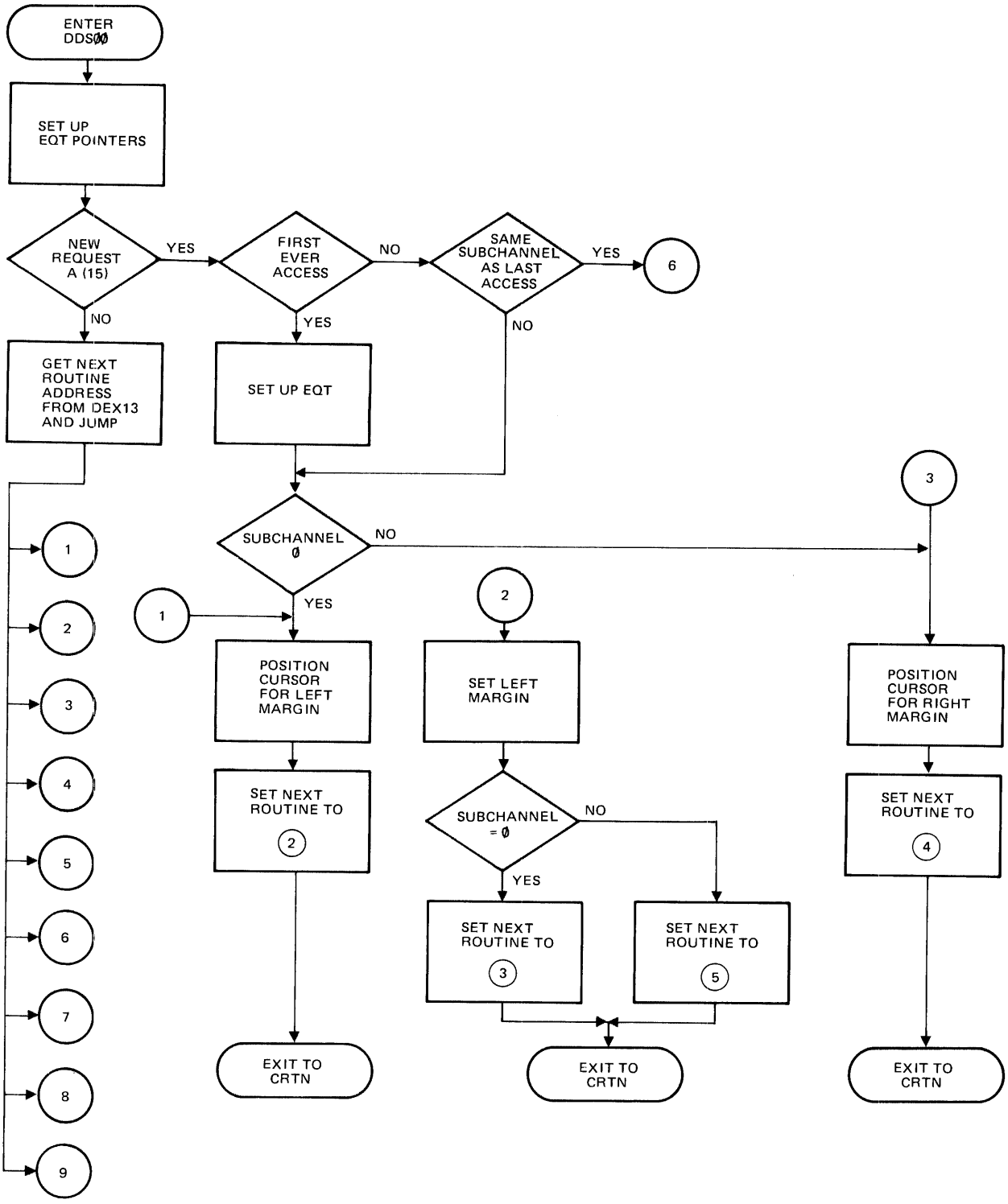
Functionally the device driver makes a series of tests on each entry to determine the action required. Processing a user request is a sequence of actions that generally fall in the following order:

1. Position left or right margin
2. Set margin
3. Position other margin
4. Set margin
5. Position cursor
6. Perform user request
7. Request cursor position
8. Read cursor position
9. Save cursor position

Each action is handled by a separate routine that saves the address of the next routine in the device driver EQT extent so that execution moves in a step by step fashion on each continuation entry to the device driver.

Operation Flow

Figure 4-1 shows the complete device driver flowchart.



7700-569

Figure 4-1. Device Driver Flow Chart

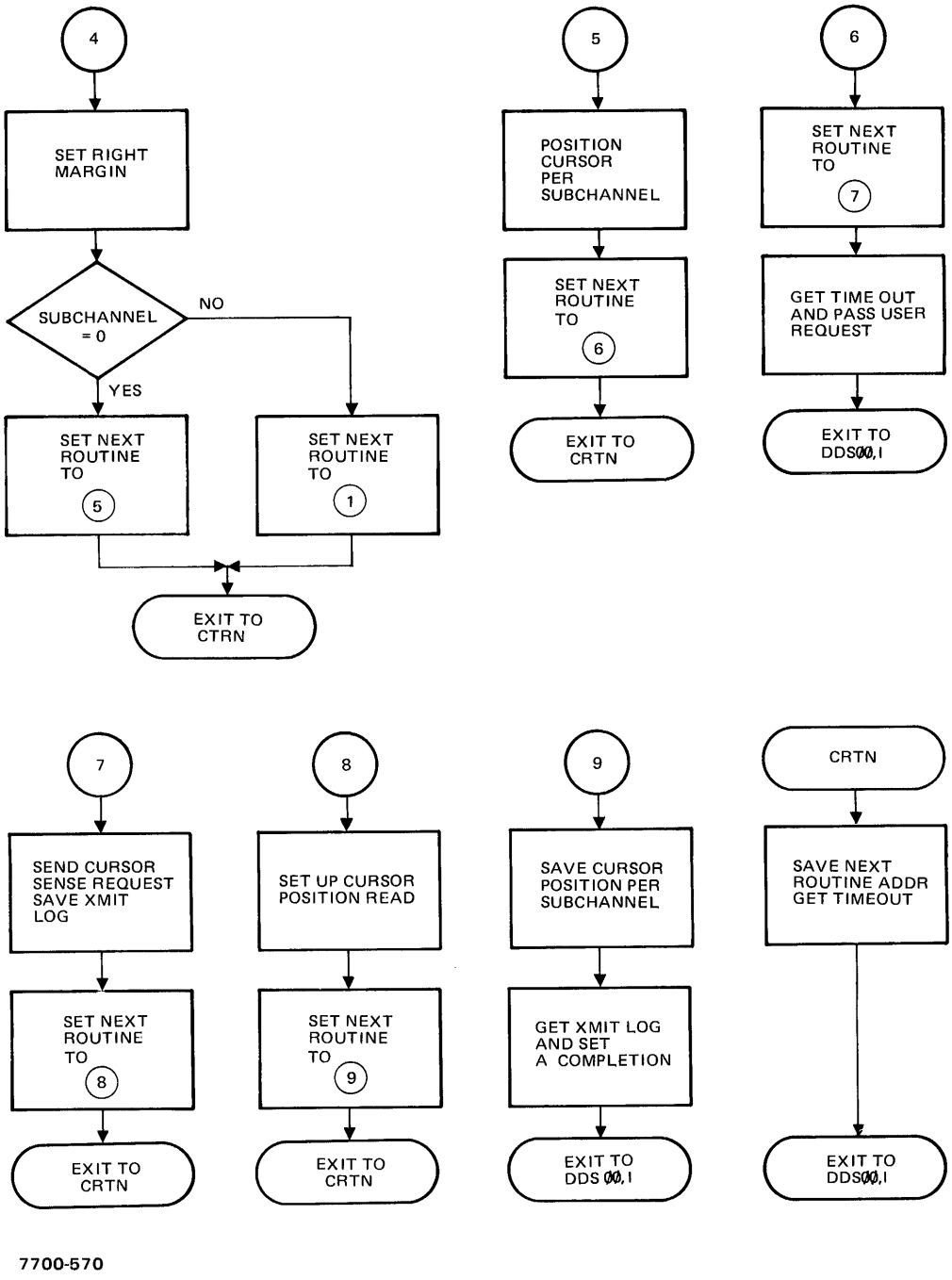


Figure 4-1 (cont'd). Device Driver Flow Chart

Set Up Device Driver EQT Extent Pointers

On each entry to the device driver the device driver EQT extent addresses are stored into a table in the device driver. On each entry the A register contains the first device driver EQT extent address and bit 15 indicates new or continuation entry. Bit 15 is saved for later testing and stripped from the address before the addresses of the device driver extent are saved.

```
    DDS00  NOP            DEVICE DRIVER ENTRY POINT
           STB TEMP       SAVE B TEMPORARILY
           CLE            SAVE A[15]
           ELA,RAR        AND CLEAR IT
           LDB DEXAD      ADDRESS OF TABLE
           LDX DM15       DECIMAL MINUS 15
*
* CREATE TABLE OF DD EXTENT ADDRESSES
*
    DEXLP  STA B,I        A CONTAINS ADDRESS OF DD EXTENT WORD
           INA            B CONTAINS ADDRESS IN TABLE
           INB            X CONTAINS COUNT DOWN
           ISX
           JMP DEXLP
            .
            .
            .
    DEXAD  DEF DEX01      ADDRESS OF TABLE
    DEX01  NOP            ADDRESS OF FIRST DD EXTENT WORD
    DEX02  NOP
            .
            .
    DEX15  NOP            ADDRESS OF DD EXTENT WORD 15
```

After the initial setup is done on each entry the device driver then tests the bit that indicates a new request entry that was saved in the E register. Continuation entries go the next routine to be executed whose address is always saved in the device driver EQT extent word 13. If it is not a continuation entry the device driver tests to see if it is the first ever entry for the EQT. Device driver EQT extension word 15 contains an ASCII "S0" if this is not the first ever entry.

EQT Setup on First Entry

On the first entry for any EQT the starting cursor positions for the left and right sides of the screen are established. The cursor positions are stored in ASCII format in the device driver EQT extent words 5 - 12. The starting positions, upper left corner of each screen, are hard coded in ASCII in the driver.

```
          .
          .
          LDA AS0        GET THE ASCII "S0"
          STA DEX15,I    SAVE IT TO INDICATE NOT FIRST ACCESS
*
          JSB .CFER      MOVE FOUR WORDS OF CURSOR POSITION
          DEF DEX05,I    TO THE DD EXTENT
          DEF S0RC       FROM THE DEFINITION LOCATION
          JSB .CFER      MOVE FOUR WORDS FOR THE OTHER CURSOR
          DEF DEX09,I    TO THE DD EXTENT
          DEF S1RC       FROM THE DEFINITION LOCATION
*
          LDA EQT5,I     GET THE EQT ENTRY WITH EQUIP TYPE
          AND TMASK      (14037B) MAKE THE TYPE 00
          STA EQT5,I     SAVE IT
          .
          .
EQT5      EQU 1644B
S0RC      ASC 4,000r000C  STARTING ROW & COLUMN SUBCHANNEL 0
S1RC      ASC 4,000r042C  STARTING ROW & COLUMN SUBCHANNEL 1
```

Subchannel Determination

On a new request entry the device driver determines what subchannel the request is addressed to. The subchannel is in the EQT word 4 bits 10-6. The subroutine GTSCH gets the subchannel and returns it in the A register.

```
GTSCH     NOP
          LDA EQT4,I     GET FROM EQT WORD 4
          AND B3.7K      (3700B) ONLY LOOK AT THE SUBCHANNEL BITS
          ALF,ALF
          RAL,RAL        POSITION TO RIGHT
          JMP GTSCH,I    RETURN
          .
          .
EQT4      EQU 1663B
```

Once the subchannel has been determined the device driver must save it in the device driver EQT extent and then go set the left or right margin. Note that although the driver specifies subchannel 0 or 1 it will use any even subchannel as the left side or any odd subchannel as the right side.

```
        JSB GTSCH      GO GET THE SUBCHANNEL FROM THE EQT
        STA DEX14,I    SAVE IT IN DD EXTENT WORD 14
        SLA            ODD OR EVEN?
        JMP RMPOS,I    ODD -- DO RIGHT FIRST
        JMP LMPOS,I    EVEN -- DO LEFT FIRST
```

Output a Setup String to the Terminal

The device driver routines to output various strings of characters to set up the terminal are all basically the same. They all set up the request in device driver EQT extent words 2-4 and exit through a common return routine. Device driver extent word 2 is set up with a read or write request code, word 3 with a buffer address inside the device driver, and word 4 is set up with the buffer length.

```
*
* POSITION CURSOR FOR LEFT MARGIN SET
*
LMPOS   DEF *+1        ADDRESS OF ROUTINE
        LDA BNWT       GET THE CONTROL WORD FOR WRITE W/O CRLF
        STA DEX02,I    PUT INTO DD EXTENT
        LDB DEX14,I    GET THE SUBCHANNEL NUMBER
        LDA LMPA       GET THE LEFT MARGIN POSITION ADDRESS POINTER
        SLB            TEST ON SUBCHANNEL
        INA            ODD -- USE THE OTHER ONE
        LDA A,I        GET THE ADDRESS OF THE CHARACTER STRING
        STA DEX03,I    PUT INTO DD EXTENT
        LDA CMLNG      GET THE CURSOR MOVE LENGTH WORD
        STA DEX04,I    PUT INTO DD EXTENT
        LDB LMSET      GET THE ADDRESS OF THE NEXT ROUTINE
        LDA MODX1      GET THE WRITE MODIFIER/EXIT COMMAND
        JMP CRTN       GO TO THE COMMON RETURN ROUTINE
         .
         .
BNWT    OCT 000102     CONTROL WORD FOR WRITE W/O CRLF
LMPA    DEF LMPAD      ADDRESS OF LEFT MARGIN POSITION TABLE
LMPAD   DEF LMPA0      EVEN SUBCHANNEL POSITION ADDRESS
        DEF LMPA1      ODD SUBCHANNEL POSITION ADDRESS
LMPA0   OCT 15446      ASCII "ESC &"
        ASC 3, 000C    COLUMN POSITION FOR LEFT SIDE LEFT MARGIN
LMPA1   OCT 15446      ASCII "ESC &"
        ASC 3, 042C    COLUMN POSITION FOR RIGHT SIDE LEFT MARGIN
CMLNG   DEC 4
MODX1   OCT 102005     WRITE MODIFIER/EXIT COMMAND:
*                      DO REQUEST IN DEX02-4, END NEXT READ ON CR
*                      DO NOT MODIFY WRITE IN DEX02
```

The common return routine is used by all of the device driver routines that initiate their own requests to the interface driver. The return routine expects the address of the next routine to be in the B register and the modifier/exit command to already be in the A register. The return routine saves the next address in the device driver EQT extent word 13 and picks up the time out value used for all setup requests.

```
CRTN    STB DEX13,I    SAVE NEXT ROUTINE ADDRESS IN DD EXT WORD 13
        LDB TO         GET TIME OUT FOR SETUP REQUESTS
        JMP DDS00,I    RETURN TO INTERFACE DRIVER
```

Stepping from one routine to the next is made simple by always saving the next routine address in device driver EQT extent word 13. On any continuation entry the device driver only has to jump through the contents of extent word 13 indirect.

```
DDS00   NOP            DEVICE DRIVER ENTRY POINT
        STB TEMP       SAVE B TEMPORARILY
        CLE
        ELA,RAR        SAVE A[15] FOR CONTINUATION TEST
         .
         .             SET UP DD EXTENT ADDRESSES
         .
        LDB TEMP       RESTORE B
        SEZ,RSS        TEST FOR CONTINUATION
        JMP CS00       IF E (WAS A[15]) IS SET, GO TO CONTINUATION
         .
         .
CS00    LDA DEX13,I    GET NEXT ROUTINE ADDRESS
        JMP A,I        GO DO IT
```

Perform the Original User Request

Since the original user request is restored to the device driver EQT extent words 2-4 on each entry to the device driver, processing the original request is quite simple. Before returning to the interface driver, the device driver only puts the system defined time out value in the B register and an exit command = 5 in the A register. In this device driver the next routine address is also saved in the device driver EQT extent word 13 to keep the flow of requests going.

```
DORQ    DEF *+1         ADDRESS OF ROUTINE
        LDB EQT14,I     GET THE SYSTEM TIME OUT WORD
        LDA SENCU       GET THE NEXT ROUTINE ADDRESS
        STA DEX13,I     SAVE IT FOR RETURN
        LDA MODX2       GET THE MODIFIER/EXIT COMMAND
        JMP DDS00,I     RETURN DIRECTLY TO THE INTERFACE DRIVER
          .
          .
MODX2   OCT 000005      UNIVERSAL DON'T MODIFY ANYTHING/DO REQUEST
*                       IN THE DD EXT MODIFIER/EXIT COMMAND
EQT14   EQU 1773B
```

Since further requests to the interface driver are required after completion of the original user request, the device driver must save the transmission log from the user request. This is accomplished by storing the contents of the B register in EQT word 8, which was the original user request length (in characters).

Read Cursor Position

SENCU senses where the cursor was left at the end of the user request. An escape - lower case a - DC1 (binary write) is sent to the terminal, requesting the terminal to send back the cursor position. The card is pre-configured for the next read by setting the high order bits in MODX1. This demonstrates the read modifier on a write request.

```
SENCU DEF *+1         ADDRESS OF THE ROUTINE
      STB EQT8,I      SAVE TRANSMISSION LOG IN EQT 8
      LDA BNWT        SET UP BINARY WRITE
      STA DEX02,I     IN DEVICE DRIVER EQT EXT
      LDA SENCA       GET ADDRESS OF CURSOR SENSE
      STA DEX03,I     FOR DD EQT EXT
      LDA SENSL       GET LENGTH FOR SENSE COMMAND
      STA DEX04,I     PUT IT IN THE DD EQT EXT
      LDA MODX1       WRITE MODIFIER/EXIT COMMAND
      LDB RDCUS       GET THE ADDRESS OF NEXT
      JMP CRTN        RETURN TO INTERFACE DRIVER
```

Final Completion Return to the Interface Driver

Completion is signified on return to the interface driver by a zero in the A register and the user request transmission log in the B register.

```
            .
            .
           LDB EQT8,I     RETRIEVE TRANSMISSION LOG
           CLA            SET COMPLETION EXIT COMMAND
           JMP DDS00,I    RETURN TO INTERFACE DRIVER
            .
            .
    EQT8   EQU 1667B
```

Device Driver Address Table

The following device driver address table is required to include the device driver with the interface driver at generation time.

```
     ASMB,Q
           NAM $DVTB,8    DEVICE DRIVER ADDRESS TABLE
           ENT $DVTB
           EXT DDS00
     *
     * DEVICE DRIVER ADDRESS TABLE
     *
     $DVTB DEC 1          NUMBER OF ENTRIES IN TABLE
           DEF DDS00+0    ADDRESS OF DEVICE DRIVER 2
           END
```

The device driver is then selected via a control 33, 000002 request to the interface on either the subchannel 0 or 1 LU.

Sample Device Driver Listing

The following sample device driver is given to illustrate the various procedures involved in interfacing to the 12792A Multiplexer Interface Driver (DVM00). It is intended as an example only and is not a supported functioning product.

```
0001                ASMB,Q
0002*
0003* MULTIPLEXER DEVICE DRIVER FOR SPLIT SCREEN OPERATION OF
0004* A 264X TERMINAL. LEFT HALF OF SCREEN IS SUBCHANNEL 0,
0005* RIGHT HALF IS SUBCHANNEL 1. FUNCTION IS OTHERWISE DVM00.
0006* THIS DEVICE DRIVER WILL SET THE MARGINS FOR LEFT OR RIGHT
0007* HALF OPERATION AND POSITION THE CURSOR TO IT'S LAST KNOWN
0008* POSITION IN THE APPROPRIATE HALF OF THE SCREEN ACCORDING
0009* TO THE SUBCHANNEL ADDRESSED ON EACH OPERATION.
0010*
0011  00000                 NAM DDS00,8 SPLIT SCREEN DEVICE DRIVER
0012                        ENT DDS00
0013                        EXT .CFER
0014*
0015  00000          A      EQU 0
0016  00001          B      EQU 1
0017  01663          EQT4   EQU 1663B
0018  01664          EQT5   EQU 1664B
0019  01667          EQT8   EQU 1667B
0020  01773          EQT14  EQU 1773B
0021*
0022  00000 000001R DEXAD  DEF DEX01
0023  00001 000000  DEX01  NOP           PHYSICAL RECORD LENGTH
0024  00002 000000  DEX02  NOP           EQT6 COPY
0025  00003 000000  DEX03  NOP           EQT7 COPY
0026  00004 000000  DEX04  NOP           EQT8 COPY
0027  00005 000000  DEX05  NOP           CURRENT SUB CH 0 ROW WORD 1
0028  00006 000000  DEX06  NOP                    "           WORD 2
0029  00007 000000  DEX07  NOP           CURRENT SUB CH 0 COL WORD 1
0030  00010 000000  DEX08  NOP                    "           WORD 2
0031  00011 000000  DEX09  NOP           CURRENT SUB CH 1 ROW WORD 1
0032  00012 000000  DEX10  NOP                    "           WORD 2
0033  00013 000000  DEX11  NOP           CURRENT SUB CH 1 COL WORD 1
0034  00014 000000  DEX12  NOP                    "           WORD 2
0035  00015 000000  DEX13  NOP           NEXT ROUTINE ADDRESS
0036  00016 000000  DEX14  NOP           CURRENT (LAST) SUBCHANNEL
0037  00017 000000  DEX15  NOP           FIRST ACCESS FLAG = ASCII S0
0038*
0039* SETUP ON EACH ENTRY
0040*
0041  00020 000000  DDS00  NOP
0042  00021 000313R        STB TEMP      SAVE B TEMPORARILY
0043  00022 000040         CLE
0044  00023 001623         ELA,RAR       SAVE A[15] FOR CONTINUE TEST
0045  00024 000000R        LDB DEXAD     SAVE EXTENT ADDRESSES
0046  00025 105745         LDX DM15
      00026 000267R
0047  00027 000001  DEXLP  STA B,I       A = ADDRESS OF DD EQT EXT 1
0048  00030 002004         INA           B = ADDRESS OF DEX01
0049  00031 006004         INB           DO FOR DEX01-15, ADDRESSES
0050  00032 105760         ISX           OF DEVICE DD EXT WORDS 1-15
```

```
0051  00033 000027R        JMP DEXLP
0052*
0053  00034 000313R        LDB TEMP       RESTORE B
0054  00035 002041         SEZ,RSS        TEST FOR CONTINUATION
0055  00036 000254R        JMP CS00       GO DO CONTINUATION
0056*
0057* SPECIAL PROCESSING TO SETUP EQT ON FIRST ACCESS
0058*
0059  00037 000017R        LDA DEX15,I    GET FIRST ACCESS FLAG
0060  00040 000277R        CPA AS0        CONTAINS ASCII "S0"
0061  00041 000250R        JMP SUBCK      NOT FIRST ACCESS, CHECK SUBCH
0062*
0063  00042 000277R        LDA AS0        FIRST ACCESS, SET UP FLAG
0064  00043 000017R        STA DEX15,I
0065*
0066  00044 000001X        JSB .CFER      SET UP SUB CH 0 ROW AND COL
0067  00045 100005R        DEF DEX05,I
0068  00046 000300R        DEF S0RC
0069  00047 000001X        JSB .CFER      SET UP SUB CH 1 ROW AND COL
0070  00050 100011R        DEF DEX09,I
0071  00051 000304R        DEF S1RC
0072*
0073  00052 001664         LDA EQT5,I     SET UP DRIVER TYPE
0074  00053 000276R        AND TMASK
0075  00054 001664         STA EQT5,I
0076*
0077* GET THE SUBCH AND DECIDE THE PATH (SUB 0 SET LEFT MARGIN FIRST,
0078* SUB 1 SET RIGHT MARGIN FIRST)
0079*
0080  00055 000261R        JSB GTSCH      GO GET SUBCHANNEL
0081  00056 000016R SVSCH  STA DEX14,I    AND SAVE
0082  00057 000010         SLA            ODD SUBCHANNEL?
0083  00060 000117R        JMP RMPOS,I    YES, DO RIGHT MARGIN FIRST
0084  00061 000062R        JMP LMPOS,I    NO, DO LEFT MARGIN FIRST
0085*
0086* POSITION FOR LEFT MARGIN SET
0087*
0088  00062 000063R LMPOS  DEF *+1        ADDRESS OF ROUTINE
0089  00063 000310R        LDA BNWT       SET UP BINARY WRITE
0090  00064 000002R        STA DEX02,I    TO POSITION THE CURSOR
0091  00065 000016R        LDB DEX14,I    GET SUBCHANNEL
0092  00066 000330R        LDA LMPA       GET LEFT MARGIN POINTER
0093  00067 004010         SLB            ODD SUBCH USE THE OTHER ONE
0094  00070 002004         INA
0095  00071 000000         LDA A,I        GET THE ADDRESS
0096  00072 000003R        STA DEX03,I    PUT IT IN THE DD EXTENT
0097  00073 000272R        LDA CMLNG      GET CURSOR MOVE LENGTH WORD
0098  00074 000004R        STA DEX04,I    PUT IT IN THE DD EXTENT
0099  00075 000100R        LDB LMSET      GET L. MARGIN SET ADDRESS
0100  00076 000314R        LDA MODX1      WRITE MODIFIER/EXIT COMMAND
0101  00077 000256R        JMP CRTN       RETURN TO INTERFACE DRIVER
```

```
0102*
0103* SET THE LEFT MARGIN
0104*
0105  00100 000101R LMSET DEF *+1        ADDRESS OF ROUTINE
0106  00101 000310R       LDA BNWT       SET UP BINARY WRITE
0107  00102 000002R       STA DEX02,I    IN DEVICE DRIVER EQT EXT
0108  00103 000332R       LDA LMSTA      GET ADDRESS OF L. MARGIN SET
0109  00104 000003R       STA DEX03,I    FOR DD EQT EXT
0110  00105 000271R       LDA MSLNG      GET LENGTH FOR MARGIN SET
0111  00106 000004R       STA DEX04,I    PUT IT IN THE DD EQT EXT
0112  00107 000016R       LDA DEX14,I    GET SUBCHANNEL
0113  00110 000010        SLA            ODD?
0114  00111 000114R       JMP *+3        YES, GO POSITION CURSOR
0115  00112 000117R       LDB RMPOS      NO, GET RIGHT MARGIN ADDRESS
0116  00113 002001        RSS
0117  00114 000154R       LDB CUPOS      GET CURSOR POSITION ADDRESS
0118  00115 000314R       LDA MODX1      WRITE MODIFIER/EXIT COMMAND
0119  00116 000256R       JMP CRTN       RETURN TO INTERFACE DRIVER
0120*
0121* POSITION CURSOR FOR RIGHT MARGIN SET
0122*
0123  00117 000120R RMPOS DEF *+1        ADDRESS OF ROUTINE
0124  00120 000310R       LDA BNWT       SET UP BINARY WRITE
0125  00121 000002R       STA DEX02,I    TO POSITION THE CURSOR
0126  00122 000016R       LDB DEX14,I    GET SUBCHANNEL
0127  00123 000345R       LDA RMPA       GET R. MARGIN ADDRESS POINTER
0128  00124 004010        SLB            ODD SUBCH USE THE OTHER ONE
0129  00125 002004        INA
0130  00126 000000        LDA A,I        GET THE ADDRESS
0131  00127 000003R       STA DEX03,I    PUT IT IN THE DD EXTENT
0132  00130 000272R       LDA CMLNG      GET CURSOR MOVE LENGTH WORD
0133  00131 000004R       STA DEX04,I    PUT IT IN THE DD EXTENT
0134  00132 000135R       LDB RMSET      GET R. MARGIN SET ADDR
0135  00133 000314R       LDA MODX1      GET WRITE MODIFIER/EXIT
0136  00134 000256R       JMP CRTN       RETURN TO INTERFACE DRIVER
0137*
0138* SET THE RIGHT MARGIN
0139*
0140  00135 000136R RMSET DEF *+1        ADDRESS OF ROUTINE
0141  00136 000310R       LDA BNWT       SET UP BINARY WRITE
0142  00137 000002R       STA DEX02,I    IN DEVICE DRIVER EQT EXT
0143  00140 000347R       LDA RMSTA      GET ADDRESS OF R. MARGIN SET
0144  00141 000003R       STA DEX03,I    FOR DD EQT EXT
0145  00142 000271R       LDA MSLNG      GET LENGTH FOR MARGIN SET
0146  00143 000004R       STA DEX04,I    PUT IT IN THE DD EQT EXT
0147  00144 000016R       LDA DEX14,I    GET THE SUBCHANNEL
0148  00145 000010        SLA            ODD?
0149  00146 000151R       JMP *+3        YES, GO SET LEFT MARGIN
0150  00147 000154R       LDB CUPOS      NO, GET CURSOR POS. ADDRESS
0151  00150 002001        RSS
0152  00151 000062R       LDB LMPOS      GET L. MARGIN POS. ADDRESS
```

```
0153  00152 000314R        LDA MODX1      WRITE MODIFIER/EXIT COMMAND
0154  00153 000256R        JMP CRTN       RETURN TO INTERFACE DRIVER
0155*
0156* POSITION CURSOR FOR OPERATION ON A SUBCHANNEL
0157*
0158  00154 000155R CUPOS  DEF *+1        ADDRESS OF ROUTINE
0159  00155 000310R        LDA BNWT       SET UP BINARY WRITE
0160  00156 000002R        STA DEX02,I    TO POSITION THE CURSOR
0161  00157 000016R        LDB DEX14,I    GET SUBCHANNEL
0162  00160 000360R        LDA CPAD       GET CURSOR ADDRESS POINTER
0163  00161 004010         SLB            ODD SUBCH USE THE OTHER ONE
0164  00162 002004         INA
0165  00163 000000         LDA A,I        GET THE ADDRESS OF LAST
0166  00164 000361R        LDB CPVAD      AND THE BUFFER FILL ADDRESS
0167  00165 105777         MVW D4         MOVE THE LAST POSITION
      00166 000272R
      00167 000000
0168  00170 000362R        LDA CPBFA      GET THE WHOLE BUFFER ADDRESS
0169  00171 000003R        STA DEX03,I    PUT IT IN THE DD EXTENT
0170  00172 000273R        LDA CPLNG      GET CURSOR MOVE LENGTH WORD
0171  00173 000004R        STA DEX04,I    PUT IT IN THE DD EXTENT
0172  00174 000177R        LDB DORQ       GET DORQ ROUTINE ADDRESS
0173  00175 000314R        LDA MODX1      WRITE MODIFIER/EXIT COMMAND
0174  00176 000256R        JMP CRTN       RETRURN TO INTERFACE DRIVER
0175*
0176* DO THE ORIGINAL USER REQUEST
0177*
0178  00177 000200R DORQ   DEF *+1        ADDRESS OF THE ROUTINE
0179  00200 001773         LDB EQT14,I    GET THE TIME OUTWORD
0180  00201 000205R        LDA SENCU      GET THE SENCU ROUTINE ADDRESS
0181  00202 000015R        STA DEX13,I    SAVE IT FOR RETURN
0182  00203 000315R        LDA MODX2      GET THE EXIT COMMAND
0183  00204 000020R        JMP DDS00,I    RETURN DIRECTLY
0184*
0185* USER REQUEST DONE, SENSE WHERE THE CURSOR WAS LEFT
0186*
0187  00205 000206R SENCU  DEF *+1        ADDRESS OF THE ROUTINE
0188  00206 001667         STB EQT8,I     SAVE TRANSMISSION LOG IN EQT8
0189  00207 000310R        LDA BNWT       SET UP BINARY WRITE
0190  00210 000002R        STA DEX02,I    IN DEVICE DRIVER EQT EXT
0191  00211 000363R        LDA SENCA      GET ADDRESS OF CURSOR SENSE
0192  00212 000003R        STA DEX03,I    FOR DD EQT EXT
0193  00213 000270R        LDA SENSL      GET LENGTH FOR SENSE COMMAND
0194  00214 000004R        STA DEX04,I    PUT IT IN THE DD EQT EXT
0195  00215 000314R        LDA MODX1      WRITE MODIFIER/EXIT COMMAND
0196  00216 000220R        LDB RDCUS      GET THE ADDRESS OF NEXT
0197  00217 000256R        JMP CRTN       RETURN TO INTERFACE DRIVER
0198*
0199* READ THE SENSED CURSOR POSITION
0200*
0201  00220 000221R RDCUS  DEF *+1        ADDRESS OF ROUTINE
```

```
0202  00221 000311R        LDA RDWCR     CONTROL WORD FOR READ W/CR
0203  00222 000002R        STA DEX02,I   PASS TO I/F DRIVER
0204  00223 000366R        LDA CRDBA     ADDRESS OF BUFFER
0205  00224 000003R        STA DEX03,I
0206  00225 000273R        LDA CPLNG     LENGTH
0207  00226 000004R        STA DEX04,I
0208  00227 000232R        LDB COMPL     ADDRESS OF NEXT ROUTINE
0209  00230 000315R        LDA MODX2     GET THE EXIT COMMAND
0210  00231 000256R        JMP CRTN      RETURN TO INTERFACE DRIVER
0211*
0212* COMPLETION ROUTINE, SAVES RETURNED CURSOR POSITION AND EXITS
0213*
0214  00232 000233R COMPL DEF *+1        ROUTINE ADDRESS
0215  00233 000016R        LDA DEX14,I   GET CURRENT SUBCHANNEL
0216  00234 000360R        LDB CPAD      GET CURSOR STORAGE POINTER
0217  00235 000010         SLA           ODD SUBCHANNEL...
0218  00236 006004         INB           YES, USE THE OTHER ONE
0219  00237 000001         LDB B,I       GET THE ADDRESS
0220  00240 005200         RBL           MAKE IT A BYTE ADDRESS
0221  00241 000375R        LDA CUBYT     GET BYTE ADDRESS
0222  00242 105765         MBT D8        MOVE BYTES
      00243 000274R
      00244 000000
0223  00245 001667         LDB EQT8,I    RETRIEVE THE TRANSMISSION LOG
0224  00246 002400         CLA           SET EXIT COMPLETION COMMAND
0225  00247 000020R        JMP DDS00,I
0226*
0227* TEST FOR SUBCHANNEL ALREADY EQUAL, SKIP MOST OF SETUP
0228*
0229  00250 000261R SUBCK JSB GTSCH     GET REQUEST SUBCHANNEL
0230  00251 000016R        CPA DEX14,I   COMPARE TO LAST ONE ADDRESSED
0231  00252 000177R        JMP DORQ,I    SAME, GO DO USER REQUEST
0232  00253 000056R        JMP SVSCH     DIFFERENT, GO SET UP TERMINAL
0233*
0234* CONTINUATION PROCESS SELECTION
0235*
0236  00254 000015R CS00  LDA DEX13,I   GET ADDRESS OF NEXT ROUTINE
0237  00255 000000         JMP A,I       GO DO IT
0238*
0239* RETURN TO INTERFACE DRIVER ROUTINE A=MODIFIER/EXIT CMD,
0240* B=NEXT ROUTINE ADDRESS
0241*
0242  00256 000015R CRTN  STB DEX13,I   SAVE NEXT ROUTINE ADDRESS
0243  00257 000312R        LDB TO        GET THE SETUP TIME OUT
0244  00260 000020R        JMP DDS00,I
0245*
0246* SUBROUTINE TO RETURN THE SUBCH IN THE A REGISTER
0247*
0248  00261 000000  GTSCH NOP
0249  00262 001663         LDA EQT4,I    GET SUBCHANNEL WORD
0250  00263 000275R        AND B3.7K     ONLY LOOK AT SUBCHANNEL BITS
```

```
0251  00264 001727          ALF,ALF
0252  00265 001222          RAL,RAL        POSITION TO RIGHT
0253  00266 000261R         JMP GTSCH,I
0254*
0255* CONSTANTS AND VARIABLES
0256*
0257  00267 177761  DM15     DEC -15
0258  00270 177775  DM3      DEC -3
0259  00271 000001  D1       DEC 1
0260  00272 000004  D4       DEC 4
0261  00273 000006  D6       DEC 6
0262  00274 000010  D8       DEC 8
0263  00275 003700  B3.7K    OCT 3700
0264  00276 014037  TMASK    OCT 14037
0265  00277 051460  AS0      ASC 1,S0
0266  00300 030060  S0RC     ASC 4,000r000C
      00301 030162
      00302 030060
      00303 030103
0267  00304 030060  S1RC     ASC 4,000r042C
      00305 030162
      00306 030064
      00307 031103
0268  00310 000102  BNWT     OCT 000102     CNWD FOR BINARY WRITE
0269  00311 000001  RDWCR    OCT 000001     CNWD FOR READ TO CR, NO ECHO
0270  00312 177160  TO       DEC -400       4 SEC TIME OUT ON SETUP
0271  00313 000000  TEMP     BSS 1
0272  00314 102005  MODX1    OCT 102005     WRITE MODIFIER/EXIT COMMAND
0273*                                       DO REQ. IN DEX02, END READS
0274*                                       ON CR, DON'T MODIFY WRITE.
0275  00315 000005  MODX2    OCT 000005     UNIV. MODIFIER/EXIT COMMAND
0276*                                       DON'T MODIFY ANYTHING,
0277*                                       JUST START REQUEST IN EXTENT
0278*
0279* TERMINAL COMMAND STRINGS
0280*
0281  00273          CPLNG   EQU D6         LENGTH OF CURSOR POSITION RQ
0282  00272          CMLNG   EQU D4         LENGTH OF MARGIN REQUESTS
0283  00271          MSLNG   EQU D1         LENGTH OF MARGIN SET REQUESTS
0284*
0285  00316 015446   LMPA0    OCT 15446      (ESC&) POS. CURSOR FOR LEFT
0286  00317 060440           ASC 3,a 000C   MARGIN SET, SUB 0
      00320 030060
      00321 030103
0287  00322 015446   LMPA1    OCT 15446      (ESC&) POS. CURSOR FOR LEFT
0288  00323 060440           ASC 3,a 042C   MARGIN SET, SUB 1
      00324 030064
      00325 031103
0289  00326 000316R  LMPAD    DEF LMPA0      ADDRESS OF LEFT MARGIN, SUB 0
0290  00327 000322R          DEF LMPA1      ADDRESS OF LEFT MARGIN, SUB 1
0291  00330 000326R  LMPA     DEF LMPAD      ADDRESS OF ADDRESSES
```

```
0292*
0293   00331 015464  LMST   OCT 15464      SET L. MARGIN COMMAND (ESC4)
0294   00332 000331R LMSTA  DEF LMST       AND ADDRESS
0295*
0296   00333 015446  RMPA0  OCT 15446      (ESC&) POSITION CURSOR FOR R.
0297   00334 060440         ASC 3,a 037C   MARGIN SET, SUB 0
       00335 030063
       00336 033503
0298   00337 015446  RMPA1  OCT 15446      (ESC&) POSITION CURSOR FOR R.
0299   00340 060440         ASC 3,a 079C   MARGIN SET, SUB 1
       00341 030067
       00342 034503
0300   00343 000333R RMPAD  DEF RMPA0      ADDRESS OF R. MARGIN, SUB 0
0301   00344 000337R        DEF RMPA1      ADDRESS OF R. MARGIN, SUB 1
0302   00345 000343R RMPA   DEF RMPAD      ADDRESS OF ADDRESSES
0303*
0304   00346 015465  RMST   OCT 15465      SET R. MARGIN COMMAND (ESC5)
0305   00347 000346R RMSTA  DEF RMST       AND ADDRESS
0306*
0307   00350 015446  CPBUF  OCT 15446      (ESC&) POSITION CURSOR FOR A
0308   00351 060440         ASC 1,a        SUB CHANNEL FOR OPERATION
0309   00352 000000  CPVAL  BSS 4          AREA TO PUT CURSOR COORD.
0310   00356 000005R CPADD  DEF DEX05      ADDR OF CURSOR COORD. SUB 0
0311   00357 000011R        DEF DEX09      ADDR OF CURSOR COORD. SUB 1
0312   00360 100356R CPAD   DEF CPADD,I    ADDRESS OF ADDRESSES
0313   00361 000352R CPVAD  DEF CPVAL      ADDRESS OF COORDINATE STORAGE
0314   00362 000350R CPBFA  DEF CPBUF      ADDRESS OF BUFFER
0315*
0316   00363 000364R SENCA  DEF SENCC      ADDR OF SENSE CURSOR COMMAND
0317   00364 015541  SENCC  OCT 15541      (ESCa) SENSE CURSOR COMMAND
0318   00365 010400         OCT 10400      DC1  SEND DATA
0319   00270         SENSL  EQU DM3        CHAR COUNT FOR SENSE COMMAND
0320*
0321* RETURN AREA FOR CURSOR SENSE
0322*
0323   00366 000367R CRDBA  DEF CRDBF      CURSOR READ BUFFER ADDRESS
0324   00367 000000  CRDBF  NOP            SPACE FOR ESC &
0325   00370 000000         NOP            SPACE FOR aY
0326   00371 000000         NOP            SPACE FOR YY
0327   00372 000000         NOP            SPACE FOR rX
0328   00373 000000         NOP            SPACE FOR XX
0329   00374 000000         NOP            SPACE FOR C
0330   00375 000761R CUBYT  DBR CRDBF+1    BYTE ADDRESS OF RIGHT BYTE
0331                        END
**  NO ERRORS *TOTAL **RTE ASMB 92067-16011**
```

Chapter 5
Device-Specific Considerations

The Multiplexer subsystem supports the following HP terminals: 2621A/P, 2626A, 2631A/B, 2635, 2675A, and 264X, as well as printer support for the 263X and the 7310A.

Other HP or non-HP devices may be used in conjunction with the Multiplexer subsystem. A prerequisite for HP support is the device must be hardwired into a point-to-point, asynchronous, bit serial environment. However, for these devices it may be necessary for the user to write simple device drivers to supplement line protocol and specific control characters.

Handshaking

Transmission and reception of data or instructions is coordinated by firmware controlled handshaking. Handshaking can be viewed from two aspects; the Multiplexer firmware or the device can act as the transmitter. Some line printers and other devices use hardware handshaking between the computer and the terminal/device; these devices are not supported.

The HP 12792A interface card uses firmware on the interface card rather than a software driver to accomplish ENQ/ACK handshaking and other line protocol necessary to communicate between the MUX card and a terminal/device. If the card configuration has handshaking enabled, data is transferred to the terminal/device in the following manner:

> The card sends data to the terminal/device in blocks of 80 characters. Between blocks an ENQ is sent and the firmware waits up to 5 seconds for an ACK. If one is received the next block is sent. If no response is given, another ENQ is sent.

If the handshaking is disabled information is transmitted serially (character by character) to the terminal/device.

The other type of handshaking is from terminal/device to the Multiplexer card and this is accomplished using DC1 an DC2 handshaking. DC1 and DC2 are used for CPU reception in block mode. This type of handshaking is controlled by the terminal driver DDV05.

DDV12 Lineprinter Driver

HP supplies a line printer driver (DDV12) similiar to DVR12, although some limitations are:

- no vertical form feed except top of form
- no over print carriage control (* in col 1)
- no status requests

DDV05 Terminal Driver

When using the HP 264X terminal in the Multiplexer subsystem, the device expects to see 8 bits/char data transfers. Users wanting to communicate with a 264X or 262X terminal with parity checking should configure the interface card for a 7 bit/char data transfer, or without parity use 8 bits/char. The 264X terminals require handshaking with the ENQ/ACK protocol in order to preserve data integrity. These terminals use a blockmode handshaking scheme with the CPU receiving DC1/DC2 protocol. A DC1 must be detected before the information can be sent across the line.

Black Box Considerations

In order to connect a "black box" RS-232-C or RS-423-A device to the HP 12792A Multiplexer Interface, the following three criteria must be examined:

RS-232-C and RS-423-A Capability

Handshaking

Driver Considerations

HP offers support to most RS-232-C and RS-423-A compatible devices. HP support is limited to correct passage of user's data to/from the user's buffer and from/to the specified Multiplexer channel data links with insertion/deletion of specifying characters and parity information.

The user should be aware of the line protocol, control sequences, and handshaking used by the device. The line protocol must match in order for two way communication to exist. The user must understand how requests are mapped in as control requests. The user must specify whether the terminal/device uses character or block mode handshaking.

The last consideration requires the user to determine if the terminal/device can function using the HP supplied drivers, or if it will require a user written device driver. Any specialized control which is required by the device not included in the user buffer indicates the need for a user written device driver. When functioning with user written device drivers, support is also limited to correct passage of EQT extent information to and from the user's device driver and the correct execution of the device drivers requests.

Dumb Devices

If the device requires no additional information beyond what is contained in the user's buffer and does not use DC1's, the device can be considered a "dumb" device and will be able to operate using DVM00 and the default device driver (device driver number one). Some devices that are normally considered dumb devices actually require CR/LF delays and will require a user written device driver for proper operation. One example of these devices is the common Teletype.

Modems

The HP 12792A Multiplexer has no modem control lines. HP will support only full duplex asynchronous modems (USA only) in the 12792A Multiplexer Subsystem. All control functions must be set on the modem.

When using modems be aware that if the modem line is disconnected, no provision is available to detect the condition. If the user was logged on under an RTE-IVB session and the line is disconnected before the user logs off, anyone dialing in to that port will be re-connected to the session in progress at the time of the previous disconnect.

Appendix A
Device Equipment Table

The HP 12792A/12828A Multiplexer Subsystem requires a Device Equipment Table (EQT) entry for each port on the multiplexer. The entry consists of 15 words plus an extension of 17 words, or a total of 32 words. The EQT entry is configured into the RTE Operating System at system generation time.

During system operation, the device and interface drivers receive channel configuration instructions, and passes information to each other through the EQT entry for that channel. Table A-1 provides the function at each word in the Equipment Table Entry in RTE-IVB and RTE-MIII operating systems.

Table A-1. Equipment Table Entry.

EQT Words 1-8:	standard in the RTE operating environment.
Word 5:	the status word, bits 7-0 describe the channel's status unless it is altered by a device driver: Bit 7: Last request timed out Bit 6: BREAK key hit Bit 5: EOT (control-D entered) Bit 4: Reserved for future use, should be set to zero Bit 3: Parity error or overflow Bit 2: Type-ahead data available Bit 1: Program scheduling enabled Bit 0: Reserved for future use, should be set to zero
Word 9:	On initiation entry this is an optional user parameter to the device driver. Thereafter, it is the starting address for transfers.

Word 10:	On initiation entry, this is an optional user parameter to the device driver. Thereafter, it is the character length of the data transfer.
Word 11:	Port Status Word 1 Bit 15: Card is busy processing a command Bit 14: Deferred abort in process (system clear request) Bit 13: Waiting for or using DCPC channel Bit 12: Buffer flush state Bit 11: Using DCPC channel 1 (select code = 6) Bit 10: I/O transfer in process Bit 9: Unsolicited interrupt in progress Bit 8: Defer abort flag Bit 7: End on CR Bit 6: End on RS Bit 5: End on CNTL D Bit 4: End on DC2 Bit 3: End on Count Bit 2: End on Character Bit 1: Edit enable Bit 0: Echo enable
Word 12:	Bit 15: This EQT is suspended on itself Bits 14-0: The address of first EQT suspended, waiting for access to the backplane.
Word 13:	Address of EQT extension
Word 14:	Standard usage: EQT time-out value reset
Word 15:	Standard usage: EQT running timer

EQT extension words: (extension word 1 = EQT word 16)

Word 16:	Address of the program to schedule -1 if none 0 if the driver has not been entered
Word 17:	Level 1 subroutine return address
Word 18:	Level 2 subroutine return address
Word 19:	Level 3 subroutine return address
Word 20:	Port ID from control 30B optional parameter, used in power fail recovery
Word 21:	Driver configuration word (from control 33B)
Word 22:	Reserved for future use
Word 23:	Length of typed-ahead data, in characters
Word 24:	Temporary, usually contains the character length of the data remaining to be transferred
Word 25:	Temporary, usually the second word of the card command

Word 26:	Temporary, usually the length of the character space left in the user buffer
Word 27:	Port Status Word 2: Bit 15: Terminating character has not yet been found Bit 14,13: 00 = control-D (terminating character) 01 = <CR> 10 = <DC2> 11 = <RS> Bits 12-9: Reserved for future use Bit 8: Port has key Bits 7-0: default status for word 5 bits 7-0
Word 28:	Device driver command to the interface driver (A-Register)
Word 29:	Device driver timeout, -10's ms
Word 30:	Device driver EXEC request
Word 31:	Device driver I/O buffer address or control requests optional parameter
Word 32:	Device driver buffer length (+words, -chars)

Any further storage used is defined by the device driver in use.

Appendix B
Device Driver Interfaces

26XX Screen Mode Device Driver

This section describes the HP supplied 26XX Screen Mode device driver DDV05 used with the Multiplexer interface driver DVM00. The driver supports HP terminals in both character and block mode. All screen mode functions (the ENTER key, soft keys, etc) are supported. Peripheral devices (CTU's, etc.) are NOT supported by this device driver.

The layout of the user I/O and Control calls are designed to be roughly compatible with DVR05. Since this subsystem is be able to support a far wider range of terminal capabilities, differences are inevitable.

For generation and initialization information please refer to the configuration guide.

DDV05 User Interface for 26XX Terminals

The device driver DDV05 utilizes the block mode read capabilities of an HP264X or 262X terminal. At boot-up time the device driver reads the terminal straps. The strappings may vary between character, block line, or block page mode. The character mode is a normal read operation, with a carriage return or CNTL-D indicating an end-of-record. A carriage return denotes an end-of-record in the block line mode, while a record separator denotes an end-of-record in the block page mode. Prior to every read request, the device driver instructs the interface driver to write a DC1 allowing the softkeys to be read.

Subchannel Assignment

No support for peripheral devices is given, therefore EQT subchannels are ignored. However, for compatibility with existing and future products, the subchannel field should be set to zero.

Control Request Definition

The control requests accepted by this driver perform various functions. Some requests require additional data which is passed to the driver through the optional EXEC parameter IPARM. Any control request not listed here is passed on to the interface driver for execution. The various requests are described as follows:

Function 11, Line Spacing

Control function 11 sends a number of CR/LF's to the terminal's display as determined by the value of the optional parameter. A maximum of 63 lines can be spaced in one request. Any value greater than 63 will be truncated modulo 64. A zero or negative line count results in one CR/LF sent.

Function 25, Update Terminal Configuration

Control function 25 (octal) causes the driver to read the strap settings on HP terminals. This information is used by the driver to assure correct terminal handshake when doing block reads, etc. on HP terminals.

A control 25 is automatically performed when the driver receives its first read request. If the terminal straps are subsequently changed manually or by escape sequences the user must issue another control 25 to keep the driver posted of any changes. Failure to do so may result in the terminal getting "hung".

Input/Output Requests

The action taken by the driver in the processing of I/O requests depends on the function code specified in the EXEC call from the user. Bits 10 through 6 of the EXEC ICNWD define the function code for the request as follows:

10 9 8 7 6	Action taken for READ request
0 X 0 X 0	input editing enabled, echo off, end transfer on <CR> or CTRL-D
0 X 1 X 0	input editing enabled, echo on end transfer on <CR>, or CTRL-D
0 X 0 X 1	input editing off, echo off, end transfer on buffer full
0 X 1 X 1	editing off, echo on, end transfer on buffer full
1 X 0 X 0	editing off, echo off, end transfer on <CR>
1 X 1 X 0	editing off, echo on, end transfer on <CR>
10 9 8 7 6	Action taken for WRITE request
0 X X X 0	end transfer on end of buffer, add CR/LF if last char in buffer is NOT "_". "_" is not printed if present as the last char in buffer
0 X X X 1	end transfer on end of buffer, nothing is added to the user's buffer
1 X X X 0	end transfer on end of buffer, nothing is added to the user's buffer

For all I/O requests note the following:

Zero length keyboard entries will not be ignored by the interface driver.

I/O transfers use a character format set up by Control request 30, and the terminal must be strapped accordingly.

Escape and Unit Separator characters are NOT stripped from the user's buffer as is done under DVR05.

Binary type transfers from the display may not be used when the terminal is in a block mode.

Read function code 3000B "program enabled block read" is not required when doing ESC-d screen reads. However, the driver is not as "forgiving" as DVR05 when terminal straps are changed without informing the driver. Control function 25 must be issued in order to prevent the terminal from being "hung".

7310 Line Printer Device Driver

Functional Overview

This section describes the HP 2631/2635/7310 Line Printer device driver DDV12 for use under the Multiplexer driver DVM00. The driver supports both normal (column 1 as carriage control) and transparent (column 1 printed) print modes. Carriage control includes Top-of-Form, single, double, and triple spacing.

For Generation and Initialization information consult the HP 12792A Multiplexer Subsystem Configuration Guide.

Write Request Processing

Write requests to the driver can be made either in a normal or transparent mode. The device driver DDV12 is used to make HP 2631/2635/7310 printers look like typical line printers to user programs. These devices use escape sequences for carriage control while standard line printers interpret the first character of each line as a carriage control character. The DDV12 device driver examines the user's first character and sends the proper escape sequence to the printer. In normal mode the first character of the line is used to direct the driver to perform carriage control. The remainder of the line is transferred to the printer. Carriage control characters recognized by the driver are:

"1" Go to Top-of-Form

"0" Space one extra line before printing (double space)

"-" Space two extra lines (triple space)

all others Single space

Transparent mode is selected by setting bit 7 of the EXEC function code. In this mode all data is shipped to the printer regardless of the data in column 1.

It should be noted that no processing of the user data is performed other than that described above. Since the printer always reacts to escape sequences and protocol characters (e.g. ENQ) the user should be careful not to place these in the user buffer.

Control Request Processing

The only control request processed by this device driver is control 11 (octal). This is used to either move the paper up (line spacing) or move the paper to top-of-form, depending on the value of the optional parameter.

IPARM > 0	move the paper up IPARM lines
IPARM = 0	move the paper up one line
IPARM < 0	go to top-of-form

The maximum number of lines that can be spaced in one request is 63. If a request is made to send more than 63 lines the value will be truncated modulo 64. (i.e 66 will send 2 lines.)

All other control requests are passed directly to the interface driver for processing. Refer to Chapter 2 for descriptions.

Appendix C
Glossary

ACK - Acknowledge	A transmission control character transmitted by a receiver as an affirmative response to the sender's block mode information.
ASCII	American Standard Code for Information Interchange.
ASYNCHRONOUS TRANSMISSION	Transmission in which time intervals between transmitted characters may be of unequal length. Transmission is controlled by start and stop elements at the beginning and end of each character.
BAUD	A unit of signaling speed equal to the number of discrete conditions or signal events per second. In asynchronous transmission, the unit of signaling speed corresponding to one unit interval per second; that is, if the duration of the unit interval is 20 milliseconds, the signaling speed is 50 baud. Baud is the same as "bits per second" only if each signal event represents exactly one bit.
BS	Backspace, Control H.
CONTROL CHARACTER	In the ASCII code, any of the 32 characters in the first two columns of the standard code table.
CR	Carriage return, Control M.
DC1 - Device control	A device control character which is primarily intended for turning on or starting a peripheral device. The host is receiving information, Control Q.
DC2 - Device control	A device control character which is primarily intended for turning on or starting a peripheral device, Control R.

DIRECT MEMORY ACCESS (DMA)	A facility that permits I/O transfers directly into or out of memory independent of the processor.
DUMB DEVICE	Device that processes one unit of information at a time. It does not contain its own local processing capability. In a smart device this is typically accomplished with a microprocessor.
DUPLEX	Simultaneous two-way independent transmission in both directions. Also referred to as full-duplex.
ECHO	A method of checking the accuracy of transmission of data in which the received data are returned to the sending end for comparison with the original data.
ENQ - Enquiry	A transmission control character used as a request for a response, Control E.
EOT	End-of-Transmission, Control D.
HALF-DUPLEX	A circuit designed for transmission in either direction but not both directions simultaneously.
INPUT EDITING	When enabled, the backspace and delete key are enabled and will affect the user's buffer. When disabled, the keys are not executed but are placed in the user's buffer.
INTERFACE	The Multiplexer card making possible interoperation between the terminal/device and the CPU.
MODEM	A device that modulates and demodulates signals transmitted over communications circuits.
PARITY CHECK	Addition of non-information bits to data, making the number of ones in each grouping of bits either always odd for odd parity or always even for even parity. This permits single error detection in each group.

PROTOCOL	A formal set of conventions governing the format and relative timing of message exchange between two communicating processes.
TRANSMISSION LOG	Length of buffer contents to or from the MUX card.
VALID TERMINATOR	End of data transfer, end of record, for example a carriage return.

A

B

C

D

E

F

H

I

L

M

N

O

P

R

S

T

U

W

READER COMMENT SHEET

12792A MULTIPLEXER SUBSYSTEM
User's Manual

12792-90002 **Update No. _____** **September 1980**
(If Applicable)

We welcome your evaluation of this manual. Your comments and suggestions help us improve our publications. Please use additional pages if necessary.

FROM:

Name _____

Company _____

Address _____

_____

_____

FOLD FOLD

NO POSTAGE
NECESSARY
IF MAILED
IN THE
UNITED STATES

BUSINESS REPLY MAIL

FIRST CLASS PERMIT NO. 141 CUPERTINO, CA.

— POSTAGE WILL BE PAID BY —

Hewlett-Packard Company
Data Systems Division
11000 Wolfe Road
Cupertino, California 95014
ATTN: Technical Marketing Dept.

FOLD FOLD

MANUAL PART NO. 12792-90002
Printed in U.S.A. September 1980

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