IDENTIFICATION

PRODUCT NAME: AC-F348E-MC

Product Name: CHQUSE0 XXDP+/SUPR USE MAN

Product Date: April 1981

Maintainer: Diagnostic Systems Engineering [illegible] System User's Manual

V1.1

[illegible] Revision: E

Diagnostic Systems Engineering

Table of Contents

PART I -- Introduction

Chapter 1: What is XXDP+? 5
Chapter 2: How to Use This Manual 7
Chapter 3: Conventions 8

PART II -- XXDP+ Monitor

Chapter 4: Brief Description of Monitor 11
Chapter 5: Starting the Monitor 12
Chapter 6: Monitor Commands 14
Chapter 7: Monitor Error Messages 18

PART III -- Runtime Services

Chapter 8: Description of Runtime Services 19
Chapter 9: Runtime Services Commands 22
Chapter 10: Runtime Services Error Messages 39

PART IV -- XXDP+ Utility Programs

Chapter 11: UPD2 42
Chapter 12: UPD1 66
Chapter 13: PATCH 76
Chapter 14: SETUP 90
Chapter 15: XTECO 92

PART V -- XXDP+ Device Drivers

Chapter 16: Devices Supported 107
Chapter 17: Driver Error Messages 108

PART VI -- Building XXDP+

Chapter 18: Monitor and Required Files 109
Chapter 19: Update Kits 110

PART VII -- Batch Control (Chaining)

Chapter 20: Introduction to Batch Control 111
Chapter 21: Batch Control of Diagnostics 116
Chapter 22: Batch Control of Utilities 117

PART VIII -- Appendices

Appendix A: XXDP+ Secondary Bootstraps 118
Appendix B: Bootstraps 118
Appendix C: Command Summary 119
Appendix D: Error Message Summary 123
Appendix E: Batch Control Summary 123
Appendix F: User Tips 124
Appendix G: Glossary 126
Appendix H: Component Names 128

Index

PART I -- Introduction

Chapter 1 What is XXDP+?

XXDP+ is the diagnostic operating system for pdp/lsi-11's. The purpose of this manual is to describe the operation of the XXDP+ System. It has been written for use by both new and experienced users. Chapter 2 contains advice on how to use this manual. The glossary (Appendix G) has definitions of terms used in this manual.

The XXDP+ System consists of four major components. These are the monitor, the diagnostic runtime services, utility programs and loadable device drivers for the utilities. The four components work together to accomplish the system functionality.

The MONITOR component is the highest level software and forms the core of the system. All of the other components require monitor support for their operation. The monitor provides program load and execution, console terminal services, batch control and file services (loading and reading files) for the system medium only. The system medium is the storage medium on the device from which the monitor was loaded. All other components utilize the terminal services for operator communications and the file services for certain operations described later.

The RUNTIME SERVICES are an extension of the monitor which provide non-diagnostic function support for certain types of diagnostic programs. These diagnostics are commonly referred to as "supervisor compatible". Among the functions that the runtime services provide are a standard operator interface, error message formatting and control of diagnostic operation.

The UTILITY PROGRAMS are used for file manipulation (e.g., moving files from one place to another), diagnostic pre-parameterization (creating diagnostic files with hardware information attached), file modification ("patching") and creation of batch control files. The utility programs use the monitor for typing and receiving messages and for loading the read/write device drivers required for file operations.

The fourth component of the XXDP+ System is the collection of device handlers, or DRIVERS. These are used by the various utility programs to access storage media and I/O devices. They are resident on system storage medium and loaded into memory as required. (The system storage medium is the medium from which XXDP+ was loaded.)

The name "XXDP+" was derived in the following manner. The characters "XX" represent the two character codes used to specify the media supported by any particular monitor. For example, DXDP+ is the XXDP+ System for the RX01, which is identified by the mnemonic ''DX''. The characters ''DP'' stand for ''Diagnostic Package'', referring to the primary function of the system: diagnostic release and distribution. The ''+'' differentiates this system from XXDP, the forerunner of XXDP+.

When ordering or discussing the various software components of XXDP+, you should always refer to the component name. A component name would be ''CHMDXA0 XXDP+ DX MONITOR BIN'' for example. The first seven characters of the name have special significance and in most cases the name of the file for a specific software component is derived from these characters. To explain the significance, let's break the word into four groups of letters: ''CH-M-DX-A0''. The first group of two letters signify the XXDP+ family of software and documentation. Only XXDP+ related software have names beginning with ''CH''. The next character defines what type of product the component is. 'M' means that this product is a monitor. Other codes represent utility programs (U), device drivers (D), runtime services (S) and manuals (Q). The third group refers to either the device supported (in the case of monitors and device drivers) or type of software (in the case of utility programs or runtime services). Appendix H has a list of the device and software type codes. The last letters refer to the revision and patch level of the component. A0 is the first version of the component. B1 would be the first revision with one patch, or temporary modification.

Examples

```
CHMDKB0 -- B revision of RK05 Monitor
CHUSUA3 -- A revision of SETUP utility with three patches
CHDDXA0 -- A revision of RX01 driver
```

For a complete list of components (without revision levels), please refer to Appendix H.

The file names for all components other than utilities are based on these names. To derive the file name from the component name, drop the ''C'' and take the next six characters as the file name. All components, except utility programs, have ''.SYS'' extensions. The utility programs are distributed under their common names, like UPD2 and XTECO, for user convenience and have ''.BIN'' (or ''.BIC) extensions.

Chapter 2 How to Use This Manual

This manual is divided into eight sections, including this introductory section. In addition to these chapters on XXDP+ and the manual, the introductory section (Part I) also contains the terminal interface and file conventions (Chapter 3). All users should familiarize themselves with this information. The next four sections describe the major components of XXDP+ in detail. The next section (Part VI) describes how to build XXDP+ on a new piece of media using any other XXDP+ media. Part VII describes the batch control features of XXDP+. The final section contains appendices with information valuable for all XXDP+ users. There is also an index for quick reference use. Users should familiarize themselves with this section of the manual.

It is recommended that the new user carefully read each section in order. Master one section before attempting the next. Each section has several examples of actual usage that will aid the reader. XXDP+ is not an overly complicated system, but a new user can have problems if he or she does not carefully read this document.

The experienced user should read over the various sections to review his knowledge of XXDP+. This manual contains information not published in previous manuals. The presentation of the material has been improved for better user understanding. The index will be of particular value to experienced users since it enables one to quickly find references to selected items.

There are two conventions used in presenting command formats throughout this manual. First, a field, or item, in a command that is a variable (such as file names) is denoted by the use of lower case characters. Fields shown in upper case characters must be entered exactly as shown. Second, optional fields in a command are enclosed in square ("[]") brackets. Sample format statement:

R filnam [addr]

In this case the operator must type "R" followed by a file name (like UPD2). The operator may also enter a starting address, but it is not required so the field is marked as optional.

One final note. This manual was prepared by and is maintained by Diagnostic Systems Engineering. Comments and suggestions are welcome. Internal users (DEC employees) should refer to the DEC phone directory listing for Diagnostic Engineering, 11-Family Systems. External users should contact their sales representative.

Chapter 3 Conventions

There are two conventions that a user must be aware of when working with XXDP+. They are the terminal interface and file conventions. This chapter describes these two conventions.

Terminal Interface

The console terminal is supported as part of the XXDP+ monitor. The console terminal driver is a simple, flag-driven handler. (Flag-driven means that no interrupts are used. This is done to prevent unsolicited interrupts from interfering with diagnostic programs.) The driver makes no distinction between upper and lower case characters. All printing characters are supported, but some characters have special significance in specific situations. The table below lists these special characters. This table briefly describes the function of the characters. Detailed descriptions and examples are left to later sections of this manual.

Char	Use
:	delimit a device specification (e.g., DK0:)
;	start comment line in a batch control file
.	beginning of a three character extension in a file name (refer to next section of this chapter)
?	''Wildcard'' character (refer to next section of this chapter)
*	''Wildcard'' specification (refer to next section of this chapter)
=	separator between input and output specifications in a command
<	equivalent to ''=''
/	command switch

In addition there are four control characters supported by the terminal driver. These are listed in the table below. Control characters depicted with an up-arrow are typed by depressing the ''CTRL'' key and then typing the designated letter at the same time.

Char	Use
^C	stop current activity and return control to program
^U	delete entire line of input so new line can be typed
^S	inhibit typing (XOFF)
^Q	resume typing (XON)

File Conventions

XXDP+ files are specified by a name and extension. The name may be up to six characters in length and the extension may be up to three characters in length. The name and extension are separated by a dot (.). Only alphanumeric characters (A-Z 0-9) may be used and spaces may not be imbedded. Sample file names:

```
UPD2.BIN
HSAAA0.SYS
TEST.1
XMONC0.LIB
```

The user would find it very inconvenient to have to type every file name in an operation involving many files or to have to get a list of every file on a medium just to get a directory of a certain type of files. To avoid this problem, XXDP+ allows the use of 'Wildcard characters''. These characters, ''?'' and ''*'', can be used as a substitute for a character or a string of characters, respectively, in a file specification. The easiest way to explain the function of wildcards is to illustrate their use with an example.

If a user wanted to obtain a list of all files on a medium that have ''.CCC'' extensions, he or she could get a directory of the entire medium and look for files with that extension. Or with the use of wildcards, the user could ask for a directory of ''*.CCC''. This would tell the file services to look for any files with ''.CCC'' extensions and any valid string of characters for a name. Other examples of the wildcard characters are:

XMON??.LIB	all files whose name starts with the characters ''XMON'' and ends with any two other valid characters (or nulls) and have the extension ''LIB''
XMON*.LIB	same effect as above
XMONC0.*	all files with the name ''XMONC0'' and any extension
XMONC0.???	same effect as above

One further observation regarding file extensions. Some extensions are used to identify particular file types. For example, batch control files have ''.CCC'' extensions. Below is a table of extensions that have particular meanings.

BIN	an executable program file that may not be run or loaded in batch control operation
BIC	an executable program file that may be run or loaded in batch control operation
SYS	a system file
CCC	a batch control file
OBJ	a DEC/X11 object module
LIB	a library file
TXT	a text file
BAK	an XTECO backup file

PART II -- XXDP+ Monitor

Chapter 4 Brief Description of Monitor

The purpose of this chapter is to give the user a general idea of how the XXDP+ monitor is constructed. If a user knows something about monitor construction, it may enable him or her to better understand monitor operation. Also, since XXDP+ is typically used in field repair situations, knowledge of what the monitor is and does may assist the technician in identifying and resolving a problem.

At load time the XXDP+ monitor is about 4K words in size and consists of three major sections: secondary bootstrap, initialization code and the runtime monitor code. The secondary bootstrap is loaded into memory at boot time and loads the remainder of the monitor into memory. Chapter 5 explains the load process in more detail. The initialization code gathers certain system information and relocates the runtime monitor. The runtime monitor is the code that is used to carry out the various operator functions described in Chapter 6. The start up process is described in more detail in the next chapter.

The runtime monitor consists of five sections: read-only device driver, console terminal driver, monitor services handler, operator interface handler and batch control handler. The read-only device driver is used for loading programs from the system medium and reading batch control files. The operator interface handler processes operator commands from the console terminal. The batch control handler processes batch files from the system medium. The services handler processes requests for monitor services which are made by utility programs via the EMT instruction. The runtime monitor is approximately 2K words in size. Since older diagnostic programs expect the size to be 1.5K, however, the monitor's lower .5K may be overwritten by a diagnostic program and then later restored by the monitor.

Chapter 5 Starting the Monitor

This chapter is divided into two sections. First, the procedure for starting the XXDP+ System is described. And second, the process that occurs when the system is starting is described. It is assumed that the reader is familiar with the peripheral device being used as the system device.

XXDP+ System Start-up Procedure

Step 1
Halt the processor (after making sure that any operating software has been "gracefully" shut down). Mount/load your XXDP+ medium. If you are working on a system that has unknown hardware problems, make sure the load device is write disabled.

Step 2
Re-enable and boot the processor. For your convenience, a number of boot procedures are listed in Appendix B.

Step 3
When the monitor has successfully loaded, it will identify itself, memory size (up to 28K words) and the drive number from which it was booted (multidrive devices only). An example is:

```
CHMDKB0 XXDP+ DK MONITOR
BOOTED VIA UNIT 0
24K UNIBUS SYSTEM
```

This is the message that would be printed after booting an RK05 monitor on a system with 24K words of memory, using drive 0. The monitor will then relocate the runtime monitor code to the top of available memory (up to 28K words) and transfer control to this code.

Step 4
The monitor will then ask you for the date. Type the date in standard format; i.e. DD-MMM-YY (day - month - year). If you wish, you may simply type a carriage return. In this case the date will be set to 01-JAN-70. After the date has been accepted, the monitor will print an address that may be used to restart the monitor in the event that the processor must be halted and restarted.

When these steps have been successfully completed, the monitor will type a dot (.) to prompt the user for commands. The user may now use any of the commands described in Chapter 6.

XXDP+ Start-up Process

The purpose of this section of the chapter is to give the user some idea of the process that occurs when he or she starts the monitor. Again, this is to assist the user in understanding and using the monitor.

When you boot the load device, the first physical block of data on the medium is loaded into the first 256(10) words of memory. This data is the secondary bootstrap. Control is passed to it from the hardware, or primary, bootstrap. The secondary bootstrap reads the remainder of the monitor from the load medium into memory. The load begins at memory location 1000(8) and ends at 20000(8) (the 4K boundary. When the load is complete, the secondary bootstrap passes control to the initialization code and the boot process is complete. If a detectable error occurs during the secondary bootstrap operation, the processor will halt. Appendix A describes these halts in detail.

If the boot process is successful, the monitor will size memory (up to 28K words), size for the presence of standard line or programmable clocks (KW-11L and KW-11P) and size for bus type. If the bus type cannot be determined, the monitor will ask the user to specify the type (UNIBUS or QBUS). The monitor will identify itself, print the unit number from which it was booted and print the system size and bus type. It will then ask for the date in DEC standard format: day-month-year (i.e., 25-DEC-80). The user may simply type a return and the date will default to 1-JAN-70. Next the monitor will print its restart address and, finally, relocate to the top of the memory (up to 28K words).

The XXDP+ monitor is designed to assume that you will operating with US-type (60 cycle) power. If you will be using European-type (50 cycle) power, you must modify the monitor. Location 1000 contains an indicator or power type, 0 for 60 cycle and 1 for 50 cycle. Chapter 11 on UPD2 explains how to modify files, but a brief example is given here.

```
R UPD2
*LOAD HMD... SYS
*MOD 1000
001000  000000   1
*SAVM DY0:
```

The underlined portion in the above example is typed by XXDP+. The user has modified the RX02 monitor and saved on the floppy diskette in drive 0, replacing the monitor already on the diskette.

Chapter 6 Monitor Commands

This chapter describes the monitor commands available to the user. These commands are summarized in Appendix C for the experienced user. The following commands are detailed in this chapter:

R	run a program
L	load a program
S	start a program
[illegible]	run a batch job (chain)
[illegible]	list directory of load medium
[illegible]	set the terminal fill count
[illegible]	enable alternative drive for system device
H	type help information
TEST	run a batch file called SYSTEM.CCC

Some commands have optional ''switches'' which consist of a single character preceeded by a ''/''. These are used to modify the command function.

Run Command

The run command is used to load and start a program stored on the load, or system, medium. (Note: the [illegible] is a combination of the Load and Start commands desc[illegible] The program must be an executable file. These a[illegible] with .BIN or .BIC extensions to their names (such as, [illegible] The format of the run command is:

R filnam[ext] [addr]

The file name must be a standard XXDP+ file name (see Chapter 3). The default extension is .BIN or .BIC. If there is a file with both extensions on the medium, the first file found will be used. After the program is found and loaded, but before the program is started, the full file name will be printed to verify the load. This is useful in determining which of possibly several programs on a medium is being run after a wildcard specification. The file will be started at the transfer address in the file (or at 200 octal in the absence of a transfer address). The operator may optionally specify a starting address. Some examples of the run command:

```
R UPD2          (load/start UPD2.BI?)
R SAMPLE.XXX    (load/start SAMPLE.XXX)
R RXDIAG 204    (load/start RXDIAG.BI?  at location 204)
```

Wildcard characters are permitted in the file specification. The first file found that fits the wildcard description will be run.

Load Command

The load command is used to load a file into memory. This command may be thought of as the first half of a run command. The program is not started. As in the run command process, the full file name of the program that was loaded is printed. All restrictions in the run command apply. The format of the load command is:

```
L filnam[.ext]
```

Some examples of the load command:

```
L DIAG              (load DIAG.BI?)
L ZDJCA2.NEW        (load ZDJCA2.NEW)
```

Start Command

The start command is used to start a file that has been previously loaded into core by a load command. No commands should be issued between a load and start command since the program loaded will most likely be overwritten. The purpose of this command sequence is to allow the user to load a program, halt the processor, modify memory contents, restart the monitor and start the program. The format of the start command is:

```
S [addr]
```

The user may optionally enter a starting address. The monitor will start the program at the transfer address in the file if the operator does not enter a starting address. The default starting address for files without specific transfer addresses is 200 (octal). Some examples of the start command:

```
L RXDIAG            (load RXDIAG.BI?)
S                   (start at transfer address)

L RXDIAG            (load RXDIAG.BI?)
S 204               (start at 204)
```

Chain Command

The chain command is used to initiate execution of a batch, or chain, file. The file must be on the system medium and must have a .CCC extension. Some batch operations accept switches. This will be detailed in documentation for the batch file. The format of the chain command is:

```
C filnam[/switches]
```

Part VII of this manual describes batch control in detail.

Directory Command

The directory command is used to obtain a list of all the files on the system medium. This list will contain five items of information: the entry number, the complete file specification (name and extension), the date the file was created, the length of the file in 256 (decimal) word blocks and the number of the first block in the file. Most files are "linked"; that is, their blocks are not in order on the medium. A few files are contiguous; that is, their blocks are in order on the medium. Contiguous files are noted in the directory by a "C" following the date.

When the directory command is given, the monitor must load the directory utility (HUDI??.SYS) which in turns requires the read/write device driver for the system medium type. These two files must be on the system medium in order for the directory command to work. If one of these files is not on the medium, the monitor will type an error message. The format of the directory command is:

D[/L][/F]

There are two optional switches for the directory command. The "/L" switch will cause the directory to be printed on a line printer rather than the console terminal. The "/F" switch causes the directory to printed in a short form. This short form only gives the entry number and file name. Ex mples of both forms are shown below.

Directory Long Form

ENTRY#	FILNAM.EXT	DATE	LENGTH	START
1	HMDKA1.SYS	02-JUN-79	12	000100
2	HDDKA0.SYS	02-JUN-79	5	000120
3	HUDIA0.SYS	02-AUG-79	6	000066

DIRECTORY SHORT FORM

```
1  HMDKA1.SYS
2  HDDKA0.SYS
3  HUDIA0.SYS
```

Fill Command

The fill count is used to control the number of fill (non-printing) characters that will be typed after a carriage return. The fill, or null, characters are typed to allow time for the carriage return before typing the next line, thus preventing overprinting. The terminals that require a fill count are: ASR, LA30, VT05 and VT50. The format of the command is:

F

The monitor will print the current fill count and then wait for the user to type the new fill count. If the user does not want to change the count, he or she should not type a number, but simply carriage return. Some examples of the fill command:

```
F
000005                    (fill count = 5)

F
000010 1                  (new count = 1)
```

The fill count is initially set to 14 in order to allow the monitor to start properly on a system that has one of the terminals that require a fill count. After start-up, the fill count is always reset to 0 since most terminals do not require fill counts. The user can immediately reset the fill count if desired.

Enable Command

The enable command is used to change the drive that the monitor considers to be the system device. For example, if the user had booted the system from drive 0 of an RK05 and later wanted to have the monitor use drive 1 as the system device (that is, as the default device), he or she could do this without re-booting the monitor by using the enable command. This command is valid for multi-drive devices only and affects drives, not controllers. The format of the command is:

E n

,where n is the new drive number.

Help Command

The help command is used to obtain a brief summary of XXDP+ commands. The contents of a file named 'HELP.TXT' are typed/printed and this file must be on the system medium. There is a switch to cause the summary to be printed on a line printer instead of the console terminal. The format of the command is:

H[/L]

Chapter 7 Monitor Error Messages

The monitor can detect the following problems and will report to the console terminal as shown. Note that all error messages begin with a question mark ("?").

Invalid Command

? INVALID COMMAND

The user has entered a command that is not recognizable to the monitor. Check your command line and retype. Monitor commands are discussed in Chapter 6 and are summarized in Appendix C.

Invalid Filename

? INVALID FILENAME

The user specified a filename in the previous command that was not in the proper format.

Bad Address

? BAD ADDR

An invalid address was specified with a run or start command.

Checksum Error

? CKERR

The monitor's driver checksums a file as it is loaded. If the computed checksum fails to match that stored with the file, the monitor reports a checksum error. You may retry the load, but if the error persists, the file image has probably been corrupted.

Lookup Error

? NOT FOUND: filnam

The specified file was not found by the monitor. There are two possibilities: 1) you spelled the file name wrong or 2) the file is not on the system medium. (Remember, the monitor can only read files that are on the system medium!) Use the directory command to get the correct spelling.

Read Error

? RD ERR

A device error occurred while reading from the system device. Retry the operation. If the error persists, suspect that the hardware may have a hard error or that the medium is bad.

PART III -- Diagnostic Runtime Services

Chapter 8 Description of DRS

The Diagnostic Runtime Services (DRS) are the part of the XXDP+ System that control compatible diagnostic programs. This program is an extension to the the XXDP+ Runtime Monitor that is automatically loaded into memory and started when a compatible diagnostic is run. DRS also provides non-test-related services, such as console terminal support, to these diagnostic programs. All diagnostic programs that are compatible with DRS share some important common features. Because of these features, they have identical structures, respond to the same general set of commands, report errors in the same way, gather hardware and operational data in the same manner and are therefore easier to use and control from both a user and system point of view.

If you are unsure of which diagnostic programs on a particular medium are DRS-compatible, you can use the SETUP utility to list these diagnostics. Chapter 14 describes SETUP.

DRS Start-up

The start-up procedure for DRS is relatively straightforward. The user issues an XXDP+ run command and the diagnostic is loaded and started. The first thing the diagnostic does is to execute an EMT instruction which transfers control back to the XXDP+ monitor. XXDP+ then loads DRS from the system medium. The DRS file is 'HSA.??', where "??" represents the revision and patch level. DRS sizes memory using memory management prior to going into command mode. If this hardware has problems, DRS will not start properly. You must run memory management diagnostics if you encounter this problem.

DRS Concepts

The user should be aware of several concepts about DRS.

CONSOLE COMMANDS - DRS uses the console terminal for communicating with the user. There are eleven commands available to the user for controlling DRS operation. (These commands are explained in the next chapter.) Unlike older-type diagnostics, DRS does not vary its operation based upon starting address or "switch registers".

COMMAND MODIFIERS (SWITCHES) AND FLAGS - The user may alter the effects of a particular command by specifying a "switch" when the command is given. For example, unless otherwise specified, most commands will affect all units (devices) that the diagnostic can test. A switch can be used to limit the effect of commands to certain units only.

UNITS - The diagnostic acts upon specified hardware. Each individual hardware "entity" is referred to as a unit-under-test (UUT) or, most commonly, as a unit. DRS is equipped to handle up to 64 units. The user refers to a unit by a number. The first unit is "0". Units are numbered according to the order in which they were specified (see HARDWARE TABLES below).

HARDWARE PARAMETER TABLES - DRS-compatible diagnostics do not autosize (determine hardware information by performing bus-related tests). The user must give the diagnostic the information about the hardware under test that is necessary. This information is stored in a set of tables called "hardware parameter tables". There is one table for each unit to be tested. The specific information required is dependent upon the diagnostic. There are some general rules that are explained in section 4 of Chapter 9. The diagnostic program will prompt the operator for the information it needs for each unit, starting with unit 0. The important concept that the user must grasp is the concept of a "table driven diagnostic" in all of the information about a hardware unit is contained in a table specific to that unit.

SOFTWARE PARAMETER TABLE - There are operational parameters that the user can select that affect the way in which a particular diagnostic will function. This information is placed into a table of data called the "software parameter table". This table (for those readers familiar with earlier processor designs) takes the place of the switch register.

PASS - A pass, or unit of diagnostic operation, is defined to be the execution of all specified tests for all active units-under-test.

TEST - DRS diagnostics are divided into independent stuctures called tests. The user may run all tests in a diagnostic or select any subset desired.

Error Messages - When a diagnostic detects an error in the device being tested, it calls upon DRS to report the error to the operator. There are three levels of error messages: header, basic and extended. The first message level supplies some general information about the error, as shown in the example below:

```
ZNAME HRD ERR 00002 ON UNIT 5 TST 012 SUB 000 PC:013134
```

The information given in the header is:

    diagnostic name - "ZNAME"
    error type - "HRD"

error number - "00002"
unit number - "5"
test number - "12"
subtest number - "0"
location of error call to DRS - "013134"

The error number is for identification and is not a running total of the number of errors that have occurred.

The basic error level is used to give a short, simple description of the error. The extended error level is typically used to give supporting information such as register contents at the time of the error. For example:

```
ZNAME HRD ERR 00002 ON UNIT 5 TST 012 SUB 000 PC:013134
REGISTER FAILED TO CLEAR AFTER BUS RESET
CSR:  000000 SCSR:  010000 ERRREG:  000000
```

The first line is the header message, the second is the basic message and the third line is the extended message. Error messages are divided into levels in order to give the operator flexibility in determining what portion(s), if any, of the error reports will be displayed or printed. (See Chapter 9, section 3, Flags.)

Chapter 9 Runtime Services Commands

This chapter is divided into four sections. Section 1 describes the DRS commands. Section 2 describes the switches, or modifiers to the commands. Section 3 describes the operational flags. Section 4 describes the table building process that the user must follow.

Section 1 -- Commands

There are eleven commands to the DRS. These commands are entered in response to the DRS prompt: DR>. The prompt is issued after the DRS is loaded, after all specified diagnostic operations are completed, after a DRS detected error, after a "halt-on-error" sequence and after DRS has been interrupted by a ^C (CTRL-C). These are tabulated below and described in the remainder of this section. The commands are grouped by related function.

Execution

START	start the diagnostic and initialize
RESTART	start diagnostic and do not initialize
CONTINUE	continue diagnostic at test that was interrupted by a ^C
PROCEED	continue from an error halt

Units Under Test

DROP	deactivate a unit
ADD	activate a unit for testing
DISPLAY	print a list of device information

Flags

FLAGS	print status of all flags
ZFLAGS	reset (clear) all flags

Statistics

PRINT	print statistical information

Exitting

EXIT	return to XXDP+ runtime monitor

The descriptions below describe the effect of each command. These effects may be modified by the use of switches that are described in the next section. Familiarize yourself with the commands before trying to use the switches. The commands may be recognized by the DRS from a minimum of three characters; thus the use of the square brackets. That is, the start command can be entered as "STA" or "STAR" or "START".

Execution Commands

STA[RT] Command

The START command starts the diagnostic from its initial state and should be the first command issued to DRS. All initialization code is executed. Refer to specific diagnostic documentation for exact nature of the initialization process carried out by a particular diagnostic. The "trap catcher" code is reloaded into the vector space. (The trap catcher is code that allows DRS to handle any unexpected interrupts and report them to the user.) The format of the command is:

STA[RT][switch-list]

where "switch-list" is any valid combination of switches (modifiers) for the START command. The switches are explained in section 2 of this chapter. The default operation of the START command is: all tests will be run on all units. All flags (section 3 of this chapter) will be cleared. The testing will continue until interrupted by the user (^C) or by a system error and an end-of-pass message will be printed after each pass. (A pass is defined to be all specified units tested once by all specified tests.)

After you issue the start command, you will be asked if you wish to change the hardware information. You must answer yes ("Y") to this question if there are no existing hardware tables. Hradware tables will already exist under three conditions. They could have been entered by a previous start command sequence, by use of the SETUP utility (Chapter 14) or by the programmer who may have hardcoded tables into the diagnostic image. Already existing tables may be overridden by the operator at this point if so desired. You will then be asked for the number of units to be tested. Enter the decimal number of units. You will then be asked for hardware-specific information for each unit according to the design of the diagnostic.

Example of START (underlined portions typed by DRS):

```
DR> STA
---
CHANGE HW (L) ?  Y
------ -- --- -
# UNITS (D) ?  n
- ----- --- -
[answer diagnostic questions]
CHANGE SW (L) ?  N
------ -- --- -
```

You will be asked for the hardware data for "n" units, where "n" is a decimal number between 1 and 64. Refer to section 4 of this chapter for assistance in answering these questions. The questions should be obvious and straightforward. If you have difficulty with the questions of a specific diagnostic, please refer to the document for that diagnostic or direct questions to Diagnostic Engineering.

After you enter all hardware data, you will be asked if you wish to change the operational data (software table). This is purely optional. You do not have to answer any software data questions unless you want to modify default diagnostic operational behavior. Section 4 of this chapter will assist you in answering the questions, but please refer to diagnostic documentation for explanations of specific questions.

If there are no hardware tables, testing will not start. You will get an error message. The following example shows what will happen.

```
DR>STA
CHANGE HW (L) ?  N
CHANGE SW (L) ?  N

NO UNITS
DR>
```

As stated above, tables may be created by 1) a dialogue with the user, 2) the SETUP utility or, in some cases, 3) by the programmer who wrote the diagnostic.

RES[TART] Command

The RESTART command, like the START command, starts the diagnostic from an initial state. The diagnostic initialization process may be different in response to a restart. Please refer to diagnostic documentation for details. The vector space is not changed. You will have the opportunity to change the contents of the software table only. The format of the command is:

RES[TART][switch-list]

where "switch-list is any valid combination of switches (modifiers) for the RESTART command. Section 2 of this chapter explains switches. The default operation of the RESTART command is: all tests will be run on all units. Flags (section 3 of this chapter) are cleared. The testing will continue until interrupted by the user (^C) or by a system error and an end-of-pass message will be printed after each pass. (A pass is defined to all specified units tested once by all specified tests.)

Sample restart (underlined portions typed by DRS):

```
DR>RES
CHANGE SW (L) ?  N
```

CON[TINUE] Command

The CONTINUE command is used to resume diagnostic operation after the user typed control-C (^C) to interrupt execution or after a halt-on-error. The diagnostic will be restarted at the beginning of the test that was interrupted, not at the first test, as would be the case with the RESTART command. The unit being tested when the diagnostic was interrupted will remain as the unit being tested. You will be given the opportunity to change the software table if you wish. You will not be able to change the hardware tables. The format of the command is:

CON[TINUE][switch-list]

where ''switch-list'' is any valid combination of switches (modifiers) for the CONTINUE command. Section 2 of this chapter describes these switches. The default operation of the CONTINUE command is: the testing will run for the number of passes remaining in the pass count specified in the last START or RESTART command. (A pass is defined to be all specified units tested once by all specified tests.) All flags will remain set/clear as previously specified.

Example of CONTINUE (underlined portions typed by DRS):

```
^C
DR>CON
CHANGE SW (L) ?  N
```

The user can also use the START and RESTART commands to resume diagnostic execution, but diagnostic initialization will take place and testing will start with the first unit, first test.

PRO[CEED] Command

The PROCEED command is used exclusively with the halt-on-error feature in DRS. When halt-on-error is in force and the diagnostic reports an error to DRS, DRS returns to command mode. The user may issue any commands at this point. The PROCEED command is special in that it restarts the diagnostic at the point where it reported the error. No initialization is done, the unit-under-test is not accessed and the vector space is unchanged. This process allows the user to examine the state of the unit being tested and then continue testing without disturbing diagnostic operation. The format of the command is:

PRO[CEED][switch-list]

where ''switch-list'' is any valid combination of switches (modifiers) for the PROCEED command. These switches are described in section 2 of this chapter. The default operation of the PROCEED command is: the flags (section 3 of this chapter) remain set/clear as specified with the previous command.

Summary of Execution Command Effects

The following is a short recap of the effects of each execution command.

START:
- trap catcher reloaded
- diagnostic initialize code executed
- user may change both hardware and software tables
- testing will start with first test on first unit
- all flags cleared

RESTART
- trap catcher not reloaded
- some diagnostic initialize code may be executed
- user may change software table only
- testing will start with first test
- flags are cleared

CONTINUE
- trap catcher not reloaded
- initialize code not executed
- user may change software table only
- testing will start at beginning of interrupted test
- flags remain in previous state

PROCEED
- trap catcher not reloaded
- user may not change hardware or software tables
- test will be resumed immediately after error call
- flags remain in previous state

Units Under Test

DRO[P] Command

The DROP command is used to deactivate a unit from testing. The unit to be deactivated must be specified using the unit switch (see section 2). All units are initially active and must be explicitly deactivated by the user or the diagnostic. The units to be deactivated must already be activated for testing. The format of the command is:

DRO[P][/UNI[TS]:n]

where ''n'' is the number of the unit to be deactivated. The unit switch is described in detail in section 2 of this chapter. The default operation of the DROP command (when the unit switch is not specified) is: all active units will be dropped from active testing.

ADD Command

The ADD command is used to activate a unit for testing. The unit switch is used to specify the unit to be activated (see section 2). All units are initially active and must be explicitly deactivated by the user or the diagnostic. The units to be activated must have already been deactivated. The format of the command is:

ADD[/UNI[TS]:n]

where ''n'' is the unit to be activated. Section 2 of this chapter describes the unit switch in detail. The default operation of the ADD command (when the unit switch is not specified) is: all deactivated units will be returned to active testing.

DIS[PLAY] Command

The DISPLAY command is used to examine the conte[illegible] hardware tables. All table data for the spec[illegible] listed on the console terminal. Units that have been [illegible] so designated. The format of the command is:

DIS[PLAY][switch-list]

where ''switch-list'' is any valid comb[illegible] (modifiers) for the DISPLAY command. [illegible] section 2 of this chapter. The default [illegible] command is: all units described in the [illegible] displayed on the console terminal.

Flags

DRS can modify diagnostic operation [illegible] several flags that the user can [illegible] described in section 3 of this [illegible] commands related to the use of these [illegible]

FLA[GS]

the FLAGS command is used to find the current status of the DRS flags. Upon receipt of this command, DRS will display the status of all flags on the console terminal. The format of the command is:

```
FLA[GS]
```

Example of FLAGS (underlined portion typed by DRS):

```
DR>FLA                        ---

FLAGS SET                          ----- ---

NONE                      ----
```

No flags are set.

```
DR>FLA                        ---

FLAGS SET                          ----- ---

IER                      ---
LOE                      ---
```

There are two flags that have been set: IER and LOE. (These flags are described in a later section.)

ZFL[AGS]

The ZFLAGS command resets all DRS flags to their cleared state. The format of the command is:

```
ZFL[AGS]
```

Section 2 -- Switches

It is assumed that the reader has read section 1 and is familiar with the DRS commands.

Switches are modifiers of command functions. For example, many DRS commands affect units. Usually a command of this type affects all units specified during hardware table build. A switch enables the user to limit the effect of the command to certain selected units. The DRS switches are:

/TESTS:test-list	execute only the tests specified
/PASS:ddddd	execute ddddd passes (ddddd = 1 to 65536)
/FLAGS:flag-list	set specified flags
/EOP:ddddd	report end-of-pass after each ddddd passes (ddddd = 1 to 65536)
/UNITS:unit-list	command will affect only specified units

All switches cannot be used with all commands. The following table shows which commands each switch may be used with.

	TESTS	PASS	FLAGS	EOP	UNITS
START	X	X	X	X	X
RESTART	X	X	X	X	X
CONTINUE		X	X	X	
PROCEED			X		
DROP					X
ADD					X
PRINT					
DISPLAY					X
FLAGS					
ZFLAGS					
EXIT					

TES[TS]

The TESTS switch is used to specify what tests will be run. The default DRS operation is to run all tests, but this switch allows the user to override the default. The format of the switch is:

/TES[TS]:test-list

The ''test-list'' is a list of test numbers separated by colons (:'s). If the test numbers are sequential, they may be specified by the first and last test number separated by a dash. For example, if tests 1, 2, 3 and 4 are to be specified, they may be entered as ''1:2:3:4'' or ''1-4''. Test numbers may be entered in any order, but tests will always be executed in numeric order.

Some examples of the TESTS switch follow. The underlined portio is typed by DRS.

```
DR>START/TESTS:5
---
DR>START/TES:1:2
---
DR>RES/TES:1:5-9:15
---
```

In the first command, the user has selected test 5 only. In the second command, the user has selected tests 1 and 2. In the final command, the user has selected tests 1, 5, 6, 7, 8, 9 and 15.

PAS[S]

The PASS switch is used to specify the number of passes that a diagnostic will run. A pass is all specified tests on all active units. Default DRS operation is "no limit" on passes. This switch allows the user to place a limit on the number of passes. The format of the switch is:

/PAS[S]:ddddd

where "ddddd" is a decimal number between 1 and 65536. Some examples of the PASS switch (the underlined portion is typed by DRS):

```
DR>STA/PASS:100
---
DR>RES/PAS:1
---
```

FLA[GS]

The FLAGS switch is used to set DRS operational flags. These flags are described in detail in the next section of this chapter. Default DRS operation is all flags cleared. The format of this switch is:

/FLA[GS]:flag-list

where "flag-list" is a list of DRS flags separated by colons (:'s). Please refer to section 3 of this chapter for detailed descriptions of flags. Some examples of the FLAGS switch (underlined portion typed by DRS):

```
DR>STA/FLAGS:LOE
---
DR>RES/FLA:LOE:IER:BOE
---
```

EOP

The EOP switch is used to specify when end-of-pass messages will be printed. These messages indicate the number of passes completed and the number of errors found. Default DRS operation is to print these messages after every pass. The format of this switch is:

/EOP:ddddd

where ''ddddd'' is a decimal number between 1 and 65536. The end-of-pass message will be printed after every ''ddddd'' passes. In the example below, the user is having the message printed after every 90 passes. (The underlined portion is typed by DRS.)

```
DR>RES/EOP:90
---
```

UNI[TS]

The UNITS switch is used to specify which available units are to be tested. Default DRS operation is to encompass all units in the scope of any command. This switch is used to limit the effects of a command to certain units. The format of the command is:

/UNI[TS]:units-list

where ''units-list'' is a list of unit numbers separated by commas. Unit numbers are decimal numbers from 1 to 64. A unit is assigned a number based upon order of entry into the tables. The first unit is unit 1. If the units are sequential, they may be specified by the first and last unit number separated by a dash (''-''). For example, units 3, 4, 5, 6 and 7 may be specified as ''3-7''. Some examples (underlined portions typed by DRS):

```
DR>DRO/UNITS:1
---
DR>ADD/UNI:2,3
---
DR>RES/UNI:5-9
---
```

In the first example, the user is asking to drop unit 1. In the second example, the user is adding units 2 and 3. And, in the last example, the user is restarting the diagnostic with units 5, 6, 7, 8 and 9 being tested.

Combining Switches

The user may specify as many valid switches, in any order, with a command as he or she desires. Simply string out the switches, one after another, on the command line. For example, if the user wanted to start a diagnostic and: 1) test units 1 through 4 only, 2) execute tests 1, 5 and 15, 3) execute 100 passes and 4) only report the end-of-pass data after every 10 passes, this is the command that would be given.

STA/UNI:1-4/TES:1:5:15/PAS:100/EOP:10

The table at the beginning of this section shows which switches are valid with each command.

Section 3 -- Flags

It is assumed that the reader has read sections 1 and 2 and is familiar with DRS commands and switches.

Flags are used to set up certain operational parameters such as looping on error. All flags are cleared at startup and remain cleared until explicitly set using the FLAGS switch. Flags are also cleared after a START or RESTART command unless set using the FLAG switch. The ZFLAGS command may also be used to clear all flags. No other commands affect the state of the flags.

Flag	Effect
HOE	halt on error - control is returned to runtime services command mode
LOE	loop on error
IER	inhibit all error reports
IBE	inhibit all error reports except first level (first level contains error type, number, PC, test and unit)
IXE	inhibit extended error reports (those called by PRINTX macro's)
PRI	direct messages to line printer
PNT	print test number as test executes
BOE	"bell" on error
UAM	unattended mode (no manual intervention)
ISR	inhibit statistical reports (does not apply to diagnostics which do not support statistical reporting)
IDR	inhibit program dropping of units
ADR	execute autodrop code
LOT	loop on test
EVL	execute evaluation (on diagnostics which have evaluation support)

HOE Flag (Halt On Error)

The HOE flag, when set, will cause DRS to execute a "halt-on-error" sequence when an error is detected by the diagnostic. Execution of this sequence does not result in an actual processor halt, but returns DRS to command mode. The exact process is:

1. When the error is reported to DRS, the error message(s) will be printed (unless printing has been inhibited).
2. DRS will return to command mode.
3. The diagnostic will have been suspended at the point of the error report to DRS and the unit being tested will be left in the state that it was in at the time of the call.

After DRS has returned to command mode, the user may issue a PROCEED command to resume diagnostic execution at the point where it was suspended. The user may also issue other commands as desired.

LOE Flag (Loop On Error)

The LOE flag, when set, will enable DRS error looping. When error looping is in effect, DRS will cause the diagnostic to continually re-execute the code that detected the error. Looping remains in effect even if the symptoms that prompted the error report dissappear. This allows for looping on intermittent errors. o stop the looping, the user must type CTRL-C (^C) to return DRS to command mode.

IER Flag (Inhibit Error Reports)

The IER flag, when set, causes DRS to inhibit all error reporting to the console terminal. While in effect, no messages will be sent to the operator except system error reports such as ILL INT (illegal interrupt) and end-of-pass reports. This feature is usually used in conjunction with error looping. It speeds up the test process and, in the case of hard copy terminals, saves paper.

IBE Flag (Inhibit Basic Errors)

The IBE flag, when set, causes DRS to inhibit a portion of error reports. There are three levels of messages in an error report. This is illustrated below:

IXE Flag (Inhibit eXtended Errors)

The IXE flag, when set, causes DRS to inhibit the extended error reporting only. The error message and basic reports will be printed.

PRI Flag (PRInter)

The PRI flag, when set, causes DRS to redirect all messages to a line printer. This does not apply to command prompts.

PNT Flag (Print Number of Test)

The PNT flag, when set, causes DRS to print the number of the test being executed.

BOE Flag (Bell On Error)

The BOE flag, when set, causes DRS to issue a ''CTRL-G'', or 'bell'' character when an error is reported by the diagnostic. This will give an audible tone at the console terminal. This feature is usually used in conjunction with the message inhibit functions.

UAM Flag (UnAttended Mode)

The UAM flag, when set, prevents the use of manual intervention during testing. Manual intervention assumes that an operator is present to undertake any necessary action. The use of this flag allows the operator to start the diagnostic and let it run unattended. When this flag is in effect, some testing will be inhibited. Refer to specific diagnostic documentation for a description of UAM flag effects in specific cases.

ISR Flag (Inhibit Statistical Reports)

The ISR flag, when set, causes DRS to inhibit the printing of statistics by the diagnostic. This is an optional feature and not all diagnostics support statistics. Consult specific diagnostic documentation to determine whether or not a diagnostic has this feature.

IDR Flag (Inhibit DRopping of units)

The IDR flag, when set, causes DRS to inhibit the dropping (deselection) of units by a diagnostic. Diagnostics may deselect a unit from the test process if an error threshold is reached or if a serious error is detected. This flag allows the user to keep the unit selected, usually for the purposes of tracing the error.

ADR Flag (autoDRop)

The ADR flag, when set, causes DRS to execute the ''autodrop'' code in a diagnostic. The purpose of this code is to test for ''device ready'' or ''device available''. If the unit being tested is not ready or available, it will be dropped (deselected). Not all diagnostics have autodrop code. Refer to specific diagnostic documentation to determine if a diagnostic does support this feature.

LOT Flag (Loop On Test)

The LOT flag, when set, causes DRS to continually execute the test(s) specified with the TEST switch. The initialize and end-of-pass code are not executed as in normal operation however.

EVL Flag (EVaLuate)

The EVL flag, when set, causes DRS to execute diagnostic evaluation code. This is an optional feature and you must refer to specific diagnostic documentation.

Section 4 -- Table Building

As stated in the beginning of the previous chapter, DRS uses hardware tables for unit information (such as register addresses, drive numbers or interrupt priority). Tables are also used for diagnostic-specific operational information (such as what data patterns to use for testing or whether or not to do read-only testing). The specific information varies from diagnostic to diagnostic, so this section only seeks to provide the user with some background information on these tables.

First, these tables must be constructed. They are constructed in three ways. One, the diagnostic is typically released (distributed) with only a "template" table for hardware data. This template contains default values for hardware information in some cases. In any event, the actual tables must be built by the user. This is most often done by starting the diagnostic with the START command and specifying the hardware data as requested by the diagnostic.

Two, the table may also be "prebuilt" by using SETUP (Chapter 14). SETUP is an XXDP+ utility program that allows the user to build tables without actually running the diagnostic. The tables are stored with the diagnostic on the XXDP+ medium and are brought into memory with the diagnostic at runtime. The user may then initiate diagnostic operation without building tables.

And three, the tables may have been built by the diagnostic programmer. These tables are already a part of the program image and may be used as they are or changed after a START command.

The operational table, which is called the software (SW) table, may not be present in all diagnostics. If it is not present, you will not be asked if you wish to change it. This is an actual storage area that has default data coded into it.

All table-related questions have the same format:

Question (type) [default] ?

The question may be something like "DRIVE NUMBER". The type is a one character code for the type of answer desired, enclosed in parenthesis. The possible types and codes are: O for octal, D for decimal, B for binary, A for ASCII and L for logical (Y or N). The question mark indicates that DRS is ready to accept the answer. If the answer is unacceptable for any reason, an error message will be typed and the user will be asked for the information again.

When you answer the hardware questions, you are building entries in a table that describes the devices under test. The simplest way to build this table is to answer all questions for each unit to be tested. If you have a multiplexed device such as a mass storage controller with several drives or a communication device with several lines, this becomes tedious since most of the answers are repetitious.

To illustrate a more efficient method, suppose you are testing a fictional device, the XY11. Suppose this device consists of a control module with eight units (sub-devices) attached to it. These units are described by the octal numbers 0 through 7. There is one hardware parameter that can vary among units called the "Q-factor". This Q-factor may be 0 or 1. Below is a simple way to build a table for one XY11 with eight units. The underlined portions are typed by DRS.

```
# UNITS (D) ?  8

UNIT 1
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  0
Q-FACTOR (O) 0 ?  1

UNIT 2
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  1
Q-FACTOR (O) 1 ?  0

UNIT 3
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  2
Q-FACTOR (O) 0 ?

UNIT 4
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  3
Q-FACTOR (O) 0 ?

UNIT 5
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  4
Q-FACTOR (O) 0 ?

UNIT 6
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  5
Q-FACTOR (O) 0 ?

UNIT 7
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  6
Q-FACTOR (O) 0 ?  1

UNIT 8
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  7
Q-FACTOR (O) 1 ?
```

Notice that the default value for the Q-factor changes when a non-default response is given. Be careful when specifying multiple units!

As you can see from the above example, the hardware parameters do not vary significantly from unit to unit. The procedure shown is not very efficient. The runtime services can take multiple unit specifications however. Let's build the same table using the multiple specification feature.

```
# UNITS (D) ?  8

UNIT 1
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  0,1
Q-FACTOR (O) 0 ?  1,0

UNIT 3
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  2-5
Q-FACTOR (O) 0 ?  0

UNIT 7
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  6,7
Q-FACTOR (O) 0 ?  1
```

As you can see in the above dialogue, the runtime services will build as many entries as it can with the information given in any one pass through the questions. In the first pass, two entries are built since two sub-devices and Q-factors were specified. The services assume that the csr address is 160000 for both since it was specified only once. In the second pass, four entries were built. This is because four sub-devices were specified. The "-" construct tells the runtime services to increment the data from the first number to the second. In this case, sub-devices 2, 3, 4 and 5 were specified. (If the sub-device were specified by addresses, the increment would be by 2 since addresses must be on an even boundary.) The csr addresses and Q-factors for the four entries are assumed to be 160000 and 0 respectively since they were only specified once. The last two units are specified in the third pass.

The whole process could have been accomplished in one pass as shown below.

```
# UNITS (D) ?  8

UNIT 1
CSR ADDRESS (O) ?  160000
SUB-DEVICE # (O) ?  0-7
Q-FACTOR (O) 0 ?  0,1,0,,,,1,1
```

As you can see from this example, null replies (commas enclosing a null field) tell the runtime services to repeat the last reply.

Chapter 10 Runtime Services Error Messages

The following are DRS error messages.

ERR HLT

A halt-on-error has occurred. DRS is now back at command mode. Halt-on-error only occurs if operator has specified this mode with a flag to DRS.

ILL INTER nnn PC nnnnnn Ps nnnnnn

An unexpected interrupt occurred through vector nnn. The Program Counter and Processor Status Word at the time of the interrupt are given.

INSUFF MEM

There is not enough memory space to store table information for the number of units that the user wants to specify.

INVAL SWTCH FOR CMND

The user specified a non-existent or non-applicable switch in the previous command. Refer to the table of switches in section 2 of Chapter 9.

INVAL UNIT

The user specified a unit that does not exist.

? LOOKUP ERROR filnam

This error message actually comes from the XXDP+ monitor. If the file name is 'HSAA??.SYS, then the diagnostic being run requires the DRS, but the DRS file is not on the system medium. Any other file name indicates that the diagnostic attempted to open a file that does not exist on the system medium.

LOOP CHNG

The range of the loop changed while looping on error was in progress.

NOT HALTED

The user attempted to enter a PROCEED command when the DRS had not executed a "halt-on-error" sequence.

NO UNIT

There are no active units. Either no units have been specified or all units have been dropped.

PASS ABORTED FOR THIS UNIT

Testing was prematurely ended for the current unit being tested. There is usually an error message from the diagnostic given prior to this message. Refer to the specific diagnostic documentation for the reason that the unit may have been aborted.

TRAP ERR AT: nnnnnn

An unrecognized TRAP instruction was executed. [illegible] TRAP instruction is used to communicate between the [illegible]tic and DRS. This error should never occur in field operation. Please report the problem if it does.

TEST # TOO BIG

The user specified a test number that is larger than the number of tests in the diagnostic program being run.

PART IV -- XXDP+ Utility Programs

There are five XXDP+ utility programs described in this section. UPD1 and UPD2 are used for manipulating files (moving files from one medium to another, loaded and saving files, etc.). The UPD1 file is smaller and more primitive than UPD2 and is used on small systems with space limitations. PATCH is used for putting temporary changes (patches) into files. SETUP is used to pre-parameterize diagnostic programs that are compatible with the runtime services. And finally, XTECO is a simple text editor used primarily for creating batch control files. There is a [illegible] DXCL, for creating DEC/X11 exercisers. This [illegible] described in this manual. Please refer to the [illegible] MANUAL for instructions on the use of DXCL.

Chapter 11 UPD2

UPD2 (Update Two) is a file manipulation utility program used for building XXDP+ media, copying files from one medium to another, deleting files from a medium, modifying files and other functions. The component name for UPD2 is "CHUP2?? UPD2 UTIL", but for the benefit of the user, the program is released under its common name: UPD2.

UPD2 runs in the lower part of memory and occupies about 6K words. It uses the runtime monitor for interfacing to the operator and loading the retrievable device drivers that it uses to accomplish device related functions. Since UPD2 requires these device drivers, the drivers you intend to use must be resident on the system medium and the system medium must be available (on-line) throughout your use of UPD2. There is one exception that is explained later in this chapter in conjunction with the DRIVER command.

This chapter is organized as follows. First there is a brief description of how to start UPD2 and a list of commands. Next there are detailed descriptions of each command, grouped and ordered as in the list. And finally there is a list of error messages with a description of the meaning of each. New users should read the entire chapter. Persons familiar with UPD2 can use the index to find the description of a particular command. The error message section can be found quickly by referring to the index under "error messages (UPD2)".

Starting UPD2

To start UPD2, type:

```
R UPD2
```

When the program has been successfully loaded by the monitor, it will type its name and a restart address and then type a prompt, "*", to tell the user that it it is ready to accept commands. The commands are tabulated below and described in detail in this chapter of the manual.

UPD2 Commands

File Manipulation

DIR	give directory of specified medium
PIP	transfer a file or files
FILE	transfer a file or files
DEL	delete a file or files
REN	rename a file

File Modification

CLR	clear UPD2 program buffer
LOAD	load a program
MOD	modify file image in memory
XFR	set transfer address
HICORE	set upper memory limit for dump
LOCORE	set lower memory limit for dump
DUMP	dump a program image

New Medium Creation

ZERO	initialize a medium
SAVM	save a monitor on a disk
SAVE	save a monitor on a tape
COPY	copy entire medium

Miscellaneous

ASG	assign a logical name to a device
DO	execute an indirect command file
READ	read a file to check validity
EOT	write logical end-of-tape mark on a tape
DRIVER	load a device driver

Returning to Monitor

BOOT	bootstrap a device
EXIT	return control to the runtime monitor

Printing

PRINT	print a file on the line printer
TYPE	type a file on the console terminal

File Manipulation

The file manipulation commands are used to maintain XXDP+ media. Files may be transferred from medium to medium, deleted from a medium or renamed. A directory, or list, of files on a medium may be obtained.

DIR

The directory command is used to obtain a directory, or list, of files on a specified medium. The user can specify where the directory is to go (console terminal, line printer or file) and what form the directory is to take (long or short). The format of the command is:

```
DIR [[devo:][ofile][/Q]=][devi:][ifile][/Q][/F][/B][/L]
```

where,

devo - output device; default is console terminal unless ofile is specified or the "=" is used, in which case the system device is default; device is assumed to be online and write-enabled

ofile - name of file for directory (devo must be a file-structured device); default is DIR.TXT; if a file already exists with the specified name, it is autodeleted

/Q - do not rewind output medium prior to beginning directory search for file with same name as ofile (for tape units only); rewind if switch not specified

devi - device from which to take directory; system device is default; device is assumed to be online and ready

ifile - files to be listed in directory; wildcards are legal; default is *.*

/Q - do not rewind input medium prior to starting directory operation (tape devices only); rewind if switch not specified

/F - give short form of directory; long form if switch not specified

/B - list number of free blocks left on input medium (random access devices only)

/L - send directory to a line printer (parallel printers only)

There are samples of both forms of the directory on page 16 of this manual. Examples of the directory command:

DIR DX0:DISK.TXT=MM0:/Q

The directory of files on MM0 will be written into a file called "DISK.TXT" on DX0. The tape will not be rewound during the operation.

DIR =DR1:*.BIN

A directory of all files with "BIN" extensions on DR1 will be written into a file called "DIR.TXT" on the system device.

DIR

A directory of all files on the system device will be typed at the console terminal.

DIR =

A directory of all files on the system device will be put into a file called "DIR.TXT" on the system device. Please take note of the effect of the equals sign on the operation of the directory command.

DIR DY0:/F/L

A short form directory of all files on DY0 will be printed on the line printer.

PIP - FILE

The PIP and FILE commands are used to transfer a single file, or multiple files, from one medium to another. There are two differences between these commands. The FILE command allows autodeletion; the PIP command does not. You may specify an output file name with the PIP command; you may not with the FILE command.

Autodeletion is simply the removal of a file from the output medium if it has the same name and extension as the file being transferred. If you attempt to transfer a file when the name already exists on the output medium using the PIP command, the file is not deleted, the transfer does not occur and a warning message is printed. If several files are being transferred, only those that do not have names and extensions that match already existing files on the output medium are transferred. If a FILE command is used, all files are transferred regardless of what files exist on the output medium and any files on the output medium that have names and extensions matching those of files being transferred are deleted prior to the transfer. The operator is not notified of autodeletions.

The PIP command may be used to rename a file during the transfer process since output file specifications are allowed. The FILE command never accepts output file specifications and files will retain their names and extensions as they are transferred.

The format of these commands are:

```
PIP [devo:][ofile][/Q]=[devi:][ifile][/Q][/N]
FILE [devo:][/Q]=[devi:][ifile][/Q][/N]
```

where,

devo - output device; system device is default; device is assumed to be online and write-enabled

ofile - file name for output file; wildcards are permitted, in which case the input files will be renamed to match the output specification (please refer to the examples below); default is *.*; if file already exists on output device, transfer will not occur; NOT USED WITH FILE COMMAND

/Q - do not rewind output medium prior to directory search for already existing file (tape devices only); rewind after each file if switch not specified

devi - input device; default is system device; device is assumed to be online and ready

ifile - input file name; wildcards are permitted; default is *.*; file(s) specified must exist

/Q - do not rewind before directory search for first file (tape devices only); rewind if switch is not specified

/N - do not type name of each file as it is found; type each name if switch is not specified

Examples of the PIP and FILE commands:

```
PIP DX0:NEW.BIN=DR1:ZZZZZZ.BIN
```

The file ''ZZZZZZ.BIN'' will be transferred from DR1 to DX0 and named ''NEW.BIN''. The transfer will not occur if ''NEW.BIN'' already exists on DX0.

```
FILE DX0:=DR1:ZZZZZZ.BIN
```

The file ''ZZZZZZ.BIN will be transferred from DR1 to DX0. It will replace a file of the same name on DX0 if such a file exists.

```
PIP =DD0:XMONC0.LIB
```

The file ''XMONC0.LIB'' will be transferred to the system device from DD0. The name will not be changed. If ''XMONC0.LIB'' already exists on the system device, the user will be given an error indication and the transfer will not occur.

    FILE =DD0:XMONC0.LIB

This command has the same effect as the command in the previous example with the exception that if "XMONC0" already exists on the system device, it will be deleted prior to the transfer.

    PIP DK1:=DK0:

All files on DK0 will be transferred to DK1. Any files that already exist on DK1 will not be transferred and the operator will be notified. The remaining files will be transferred.

    FILE DK1:=DK0:

In this case all files on DK0 will be copied to DK1, regardless of what files already exist on DK1. This command provides a convenient method of putting updated files onto an existing XXDP+ medium.

    PIP MM0:FILE??.*=

All files on the system device will be transferred to MM0 and RENAMED to have the characters "FILE" replace the first four characters of the original name. Be careful if you use wildcards on the output specification!

DEL

The DELETE command is used to remove a file, or files, from an XXDP+ medium. The actual process for deletion is to remove the file name from the directory and deallocate the physical blocks used by the file. The format of the command is:

    DEL dev:ifile[/N][/Q]

where,

dev - device where file resides; device is assumed to be online and write-enabled; there is no default

ifile - file(s) to be deleted; file must be on device specified; extension must be specified (unless file has no extension); wildcards are accepted

/N - inhibit printing of file names as they are deleted; if switch is not specified, names will be printed

/Q - do not rewind before searching for file(s) (tape devices only); if switch is not specified, tape will be rewound prior to searching for each file to be deleted

Examples of the delete command:

```
DEL MM0:XYZ001.TXT/Q
```

The file named "XYZ001.TXT" on MM0 will be deleted. The search for the file will begin at the point where the tape is currently positioned (the no rewind switch has been specified). An error will be reported if the device is not online and write-enabled or if the file does not exist.

```
DEL *.OLD
```

All files on the system device with "OLD" extensions will be deleted. An error will be reported if the device is not online and write-enabled or if there are no filed with "OLD" extensions.

REN

The rename command is used to change the file specification of an existing file without doing a transfer. The name of the file as recorded in the directory is changed, but there is no movement of data. The format of the command is:

```
REN [dev:]newnam=[dev:]oldnam
```

where,

- dev - device where file to be renamed exists; default is system device; device is assumed to be online and write-enabled; device must be same for both input and output

- newnam - new file specification; file with same specification must not exist on device; wildcards are accepted

- oldnam - current file specification; file must exist on device; wildcards are accepted

Examples of the rename command:

```
REN DX1:DIAG.OLD=DX1:DIAG.BIN
```

File Modification

One of the prime functions of UPD2 is the modification of binary files. When a diagnostic program is found to have a deficiency, one of the corrective measures taken is to issue a DEPO or "patch order". This is a temporary change to a released program. UPD2 is one of the means of implementing these temporary remedies. The program in question is loaded from an XXDP+ medium into an area in memory called the program buffer. This area lies in the physical memory space between the monitor and UPD2 and its size is determined by the amount of memory in the system. The size is equal to the system size minus 8K words, but no larger than 20K words. The program image, now resident in memory, may be modified and then put back onto an XXDP+ medium ("dumped"). The transfer address and load image size may also be altered by the user at this time.

The next seven commands relate to this function of UPD2. In the descriptions of these commands, locations within a program that has been loaded into the program buffer are referred to as "virtual locations" since their addresses are relative to the first physical location in the program buffer and not the first physical memory location as would be the case if the program had been loaded by the monitor.

The location addresses given in a DEPO (Diagnostic Engineering Patch Order) are treated as virtual when using UPD2. For example, if the DEPO says to modify location 1002, that will be the virtual location you will refer to in the MOD command. If the program has been loaded by the monitor however, 1002 would be an absolute address in memory, not a relative location in the program buffer.

The file modification process applies to image (BIC or BIN) files only. The process, briefly, consists of the following steps:

1. Load the file into the program buffer with the LOAD command.
2. Change the size of the image, if necessary, with the HICORE and LOCORE commands.
3. Modify the contents of the desired location(s) with the MOD command.
4. Modify the transfer address, if desired, with the XFR command.
5. Write the image onto media with the DUMP command.

These commands are detailed in the following sections. A sample file modification is given after the descriptions.

CLR

The clear command clears the program buffer into which programs are loaded for modification. This command allows the user to assure that unused locations in a program are set to zero when the program image is dumped to a medium. The format of the command is:

CLR

LOAD

The load command is used to load a binary file into memory for the purpose of modifying the program image. As the file is loaded, a checksum is computed and compared with a checksum stored with the file. The command format is:

LOAD [devi:]ifile[/N][/Q]

where,

devi - device from which to load the file; default is the system device; device is assumed to be online and ready

ifile - file to be loaded; no default accepted

/N - inhibit printing of upper and lower memory limits and file name found (if wildcard used)

/Q - inhibit rewind before searching for file (tapes only)

Wildcards are permitted in the file specification, but the user should take care when doing so. The data in the program buffer will be the result of the overlays of each file found. The program buffer is not cleared between loads and the unused locations for the last file loaded will not necessarily contain zero's for contents. The wildcard feature is really only useful for doing file sanity checks where the user is interested in verifying that files are not corrupted.

After the load has been successfully completed, UPD2 will print the transfer address and core limits (the lowest and highest virtual memory locations used by the program). These parameters may be altered by the user as described in subsequent command descriptions. Example:

```
LOAD DX0:PROG1.BIN

XFR: 000001   CORE: 000000,020000
```

This command is also used to load an XXDP+ monitor image into the program buffer. This must be done as part of the media build process. The section on New Medium Creation in this chapter and Part VI of this manual describe this process.

MOD

The modify command is used to alter the contents of one or more virtual memory locations in a program that has been loaded by UPD2. The format of the command is:

MOD nnnnnn

,where nnnnnn is the octal address of the virtual memory location where the contents are to be modified. UPD2 will respond by typing the address and the current contents. It will then wait for the user to either type new contents or accept the the current contents by typing a "carriage return". If the user wishes to modify, or examine, two or more consecutive virtual memory locations, he or she should type a "line feed" after modifying each location. Examples:

```
MOD 2460
002460 770  771(CR)
```

The user has modified virtual memory location 2460.

```
MOD 12004
012004 012736(CR)
```

The user has examined, but not modified, virtual location 12004.

```
MOD 1220
001220 120  167(LF)
001222 120  1234(CR)
```

The user has modified two consecutive virtual memory locations (1220 and 1222).

XFR

This command is used to modify the transfer address in the program that has been loaded. The file created by the DUMP command will have this transfer address. This is address of the location to which control will be transferred when the program is started.

After the user has entered this command, UPD2 will print the current transfer address. If the user does not wish to alter the transfer address, he or she should immediately type a "carriage return". The format of the command is:

XFR

Examples of this command:

XFR
000200 001000

The user has changed the transfer address from 200 (octal) to 1000 (octal).

XFR
002000(CR)

The user has not altered the transfer address.

HICORE

The high core command is used to alter the address of the highest virtual memory location that will be transferred during a "dump" operation. The default location is printed after the LOAD command. The format of this command is:

HICORE

The address of the highest virtual memory location to be used during a dump operation will be printed. UPD2 will then wait for the user to alter or accept the location. The user may enter a new location by typing the octal address or accept the location as is by typing a "carriage return". This address must be above that of the lowest virtual location (low core) and below that of the top of the program buffer.

Examples:

HICORE
40000 45000

The user has changed the upper virtual location from 40000 to 45000.

HICORE
100000 (CR)

The user has allowed the upper location to remain as 100000.

LOCORE

The low core command is used to alter the address of the lowest virtual memory location that will be transferred in a dump operation. The default location is printed after the load operation has been completed. The format of this command is:

LOCORE

Examples:

```
LOCORE
000200    0
```

The user has modified the low memory address to zero.

```
LOCORE
000000    20
```

The user has [illegible] the low memory address to 20 (octal).

DUMP

The [illegible] used to write the program image in the pro[illegible] a file on a medium. The image size is det[illegible] upper and lower memory limits displayed by the HICORE [illegible] commands. A transfer address will be put into the file. [illegible] address can be examined and altered with the XFR command. The format of this command is:

```
DUMP [dev:]ofile[/Q]
```

where,

- dev - device to which file is to written; default is system device; device is assumed to online and write-enabled
- ofile - file name for binary file; wildcards are not accepted; file with specified name must not already exist on device
- /Q - inhibit rewind before searching for logical end-of-tape (tapes only)

Examples of the dump command:

```
DUMP DY0:ZRLAA1.BIN
```

The program image will be written to DY0 and given the file name: 'ZRLAA1.BIN'.

```
DUMP FILE3.BIC
```

The image will be written to the system device and given the specified name.

Sample Modification

The following sample file modification illustrates the use of the preceeding se[illegible] commands. It is based on the following patch for a fictitio[illegible] diagnostic named ZXXXB0.BIN:

Location	Old Contents	New Contents
1224	106701	240
1226	177660	240
1452	376	374

The UPD2-user dialogue used to accomplish this is shown below. The underlined portion is that typed by UPD2. (LF) and (CR) refer to line feed and return.

```
.R UPD2

CHUP2B0 XXDP+ UPD2 UTILITY

RESTART ADDRESS:  002432

*LOAD ZXXXB0.BIN

XFR:  000001       CORE:  000200, 027230

*MOD 1224
001224   106701     240(LF)
001226   177660     240(CR)

*MOD 1224
001224   000240     (LF)
001226   000240     (CR)

*MOD 1452
001452   000376     374(CR)

*MOD 1452
001452   000374     (CR)

*DUMP ZXXXB1.BIN

*
```

The user examines locations previously modified in order to verify the changes. The user also renames the file to reflect the patch level.

New Medium Creation

The commands described in this section are used to create new XXDP+ media. The build process is described in detail in Part VI of this manual. The user should familiarize him/herself with these commands before proceeding to Part VI.

ZERO

The zero command initializes a medium by clearing the bit map (random access devices) or writing an end-of-tape mark (sequential access devices) and placing an empty directory on the medium. CAUTION: all data on the medium prior to a zero operation is irretrievably lost after the operation. UPD2 makes no attempt to determine what is on a medium and will destroy customer data. A warning is printed whenever this command is invoked, stating which device is involved. The user must then verify that the zero operation is to take place. The format of the command is:

```
ZERO dev:
```

There is no default for the device. The device must be online and write-enabled. The following warning will be issued after the command is entered:

```
USER DATA ON dev WILL BE DESTROYED!
PROCEED?  (Y/N/CR=N)
```

The only answer that will confirm the user's intent to carry on is 'Y'.

There will be an additional warning message if you specify the system device in the ZERO command.

```
ZERO SYSTEM DEVICE
YOU MAY NEED AN ADDITIONAL DBIVER
PROCEED?  (Y/N/CR=N)
```

If you wish to proceed with the process, you must type a 'Y'. The meaning of the warning is that you must assure the presence in memory of two drivers, one for the system device and one for the device from which files will be moved to the new media in the system device. To assure that the necessary drivers are in memory, use the DRIVER command.

SAVE - SAVM

The save commands are used to place a bootable monitor on a medium. The SAVE command is used for sequential access devices and the SAVM for random access devices. (Sequential access devices are MM, MT, MS and CT.) The command places the appropriate secondary bootstrap on the boot block and places the monitor file on the medium in a predetermined section. The monitor image file (e.g.; HMDKBO.SYS) must have been loaded into the program buffer using the load command. See example below. Refer to Chapters 18 and 19 for details on building XXDP+ media.

The medium need not be initialized with the zero command if the medium is already XXDP+ compatible. In this case only the monitor and bootstrap on the medium prior to the save operation are lost. All other files are preserved.

The format of the commands are:

```
SAVE dev:
SAVM dev:
```

where dev: is the device with the medium that is to be made bootable. The device must be online and write-enabled. The device must be sequential access (e.g.; magtape) when using the SAVE command. It must be random access (e.g.; disk) when using the SAVM command.

Example:

```
LOAD HMDX??.SYS
SAVM DX1:
```

COPY

The copy command is used to copy the entire contents of one medium to another identical medium (e.g.; RP06 to RP06). The copy process can take two forms. The first is an ''image copy''. This is a block-for-block transfer and is very fast since all available memory is used as a buffer. If the device is a bad block device, there is a chance that a bad sector may be encountered during the copy process. In this case the process will be aborted. You will then have to use the second form, ''file copy''. This is slower since only one block is transferred at a time. In both cases, the former contents of the medium are destroyed. There is no check for medium type and customer data could be lost. Be careful! There will be a warning message as shown below. The copy will proceed only if a '"Y"' is typed. The format of the command is:

```
COPY devo:=devi:
USER DATA ON devo WILL BE DESTROYED!
PROCEED?  (Y/N/CR=N)
```

where devo and devi are the output and input devices respectively. The two devices must be the same type. Both devices must be online and the output device must be write-enabled.

Miscellaneous

ASG

The assign command is used to assign a logical unit number to a device. The device can then be referenced by this number in ensuing UPD2 commands. The format of the command is:

```
ASG dev:=n
```

where,
  dev - device to be assigned

  n - logical unit number (0-7)

The primary use for the assign command is to facilitate the use of the DO command and indirect command files (see next section).

Example of assign command:

```
ASG DK0:=0
PIP 0:=DK1.*.CCC
FILE 0:=MM0:FILE.NEW
```

DO

The do command is used to execute an indirect command file for UPD2. This file is a text file that contains one or more commands executable by UPD2 with the exception of EXIT. The user may create a command file that accomplishes some common task such as building new media. This saves time and effort on the part of the user since he or she need not enter each command by hand. The format of the command is:

```
DO file.ext
```

The specified file must be on the system device, therefore you cannot specify a device.

There are two functions available in the indirect command file in addition to the normal set of UPD2 commands. First, any command line beginning with a semicolon (;) will be treated as a comment. That is, no action will be taken; the line is merely printed. Second, a command line beginning with a dollar sign ($) will also be treated as a comment, except the processing of the command file will cease after the line is printed and resumed when a "Control X" is typed. (Control X is typed by depressing the X key while holding the CTRL down.) This second function can be used to stop activity while the operator performs some required task such as mounting a new medium or placing a device online.

The file can be made more global in scope by using logical unit numbers instead of device names in the commands. The user can then assign logical unit numbers prior to using the indirect command file using the assign command. The example below illustrates the combined use of the two commands.

```
Sample Command File:  RMBLD.TXT
  ZERO 1:
  LOAD 0:HMDR??.SYS
  SAVM 1:
  FILE 1:=0:*.SYS
  FILE 1:=0:UPD2.BIN
```

The above file can be used to build the XXDP+ System on any RM02/3 using any other XXDP+ medium. (Note that the command line containing a ''Y'' only is required to verify the zero process. See the zero command description.) The process for doing this is:

```
.R UPD2
*ASG DR2:=1
*ASG MT0:=0
*DO RMBLD.TXT
*EXIT
```

The underlined portion of the above example is that typed by UPD2. The remainder is typed by the user.

READ

The read command is used to check device and media integrity. Each block of the file specified in the command is read into memory and a checksum is calculated. The computed checksum is compared to the checksum stored with the file. The format of the command is:

READ [dev:]ifile[/N][/Q]

where,

dev - device from which file(s) are to be read;default is system device; device must be online

ifile - file(s) to be read; wildcards are accepted

/N - do not print name of each file as it is read

/Q - do not rewind before searching for specified files (tape devices only); rewind will occur if switch is not specified

EOT

The end-of-tape command is used to place a logical end-of-tape marker on a tape at the current position. Note, the tape is not rewound. All files after the current position will no longer be accessible. The marker consists of two consecutive tape marks. Any data beyond this point on the tape is lost. The format of the command is:

EOT dev:

The device must be a tape unit. The system device, if a tape unit, is the default device.

DRIVER

The driver command is used to explicitly load a read/write device driver into memory. Up to two drivers may be loaded. If a third driver is loaded, one of the drivers currently in memory will be lost. If a requested driver is already in memory, no action is taken. The format of the command is:

DRIVER driver[/driver]

where ''driver'' is the two character device name (e.g.; DX = RX01). The list of supported devices and their names is in Chapter 16. Note that two devices may be specified with one command.

DRIVER DX:/DK:

The purpose of the driver command is to allow a user to build XXDP+ media with limited resources. If the system device is required for building a new medium, the user can load the drivers required, remove the system medium, mount the new medium and build XXDP+.

Returning to Monitor

There are two commands that allow the user to return control to the monitor. These are described below.

BOOT

The bootstrap command is used to start the monitor in the same manner as the hardware bootstrap. The purpose of this command is to allow the user to boot a device other than the original system device. The booted device is now the system device. The format of the command is:

```
BOOT dev:
```

The device must have a bootable medium mounted. The boot process consists of loading the first physical block (boot block) into the first 256 (decimal) words of memory and starting execution at location 0.

EXIT

The exit command is used to return control to the runtime monitor. The format of the command is:

```
EXIT
```

Printing

There are two commands to output textual information from files.

PRINT

The print command is used to read textual information from a file and output it to a line printer. The file must contain text in ASCII format. The format of the command is:

```
PRINT [dev:]ifile[/Q]
```

where,
  dev - device where file is located; default is system device; device must be online

  ifile - file to be printed; must exist and contain text

  /Q - do not rewind before searching for specified file (tape devices only); rewind will occur if switch is not specified

Example:

```
PRINT MT1:HELP.TXT/Q
```

The file "HELP.TXT" will be read from the tape on MT0 and printed on the line printer. The search for the file will begin at the current tape location; no rewind will occur.

TYPE

The type command is used to read textual information from a file and output it to the console terminal. The file must contain text in ASCII format. The format of the command is:

```
TYPE [dev:]ifile[/Q]
```

where,
  dev - device where file is located; default is system device; device must be online

  ifile - file to be typed; must exist and contain text

  /Q - do not rewind before searching for specified file (tape devices only); rewind will occur if switch is not specified

Example:

```
TYPE SYSTEM.CCC
```

The file "SYSTEM.CCC" will be read from the system device and printed on the console terminal.

Error Messages

The error messages that could be typed during UPD2 operation are listed alphabetically below. The lower case letters ''device error'' refer to a series of specific errors that can be detected by the device drivers. The possible errors are listed after the error messages.

? CHECKSUM ERROR

Each block in a binary file has a checksum stored in it. If the checksum calculated while reading the block does not match the checksum in the block, this error will be printed. Try the operation again, but the file was probably corrupted.

? device error

Each read/write device driver may detect an error during an operation. The driver will report the type of error and return control to the program being used. This program will append any additional information as shown in the next six error messages. The errors that can be reported by drivers are: READ ERROR and WRITE ERROR.

? device error ON INPUT DEVICE

The specified error occurred on the input device specified in the last operator command. If the error persists, the media or the hardware may be bad. Try running diagnostics for the specific device.

? device error ON OUTPUT DEVICE

Same as above, except output device has problem.

? device error ON INPUT DEVICE DIRECTORY

The specified error occurred while accessing the directory on the input device specified in the last operator command. There may be problems with either the device or the media. Try running diagnostics for the device.

? device error ON OUTPUT DEVICE DIRECTORY

Same as above, except the output device specified in the last operator command is involved.

? device error WHILE LOADING DRIVER FOR dev

The specified error occurred while the driver for the specified device was being loaded into memory.

? device error WHILE READING filename

The specified error occurred while the specified file was being read.

? DEVICE FULL

The capacity of the output device has been exceeded. For disk devices (random access), there are not enough physical blocks remaining to store the file. Any blocks allocated during the attempt to write the file are deallocated. For tape devices (sequential access), the physical end-of-tape mark was reached while the file was being written. In both cases, no file is created. Delete some existing files or use another medium.

? DIRECTORY FULL

There are no remaining entries in the directory and the name of the file and other data cannot be entered. No file is created. Delete some existing files or use another medium.

? UNEXPECTED END-OF-FILE

The logical end of a file was encountered before it was expected. The file in question is corrupt.

? FILE ALREADY EXISTS

The name of the file specified for output matches that of a file that already exists on the output medium. Delete the old file or use a different name.

? INVALID ADDRESS

There are three operations that can cause this error. One, when using the MOD command to modify a virtual location, the address of the location given was odd or not within the upper and lower core limits. Two, the address given in a LOCORE command was higher than the current high core limit. And, three, the address given in a HICORE command was lower than the current low core limit.

? INVALID COMMAND

The last command specified was not one of the legal commands for UPD2. Re-enter the command properly.

? INVALID DEVICE

This error has a number of causes, depending on the command being used. For file related commands (DIR, COPY, ZERO, SAVE, SAVM, DEL, BOOT and EOT), one of the devices specified is not file-structured (like paper tape). For EOT, the device is a non-tape device. For COPY, the specified devices are not identical types. For SAVE, the device is not a sequential access device. For SAVM, the device was not a random access device.

? INVALID FILENAME

The filename specified in the last command was invalid. Check the command and re-enter properly.

? INVALID NUMBER

A number specified in the last command was not entered properly. Possible problems are: not a number (e.g., 12e4) or not proper radix (e.g., 1292 is not octal).

? INVALID SWITCH

The last command was entered with a switch that is not recognized by UPD2. Re-enter the command properly.

? LOGICAL DEVICE NOT ASSIGNED

An attempt was made to use a logical unit number without first assigning it to a device. (See ASG command.)

? NOT FOUND

The file specified for input in the last command was not found on the device specified.

? NOT FOUND: HDxx??SYS

The driver for device "xx" was not found on the system medium. This message is printed by the monitor driver which is used to load device drivers for UPD2. Transfer the required driver file to the system medium.

? SPECIFY DEVICE

The last command specified does not allow the use of default device specification. Re-enter the command with the device(s) explicitly specified.

? OVERFLOW

An attempt was made to load too large a program into the program buffer.

? SYNTAX ERROR

The last command was entered improperly. Re-enter the command properly.

Chapter 12 UPD1

UPD1 (Update One) is a file modification program that duplicates some of the functions in UPD2 (Chapter 11). It is used exclusively for modifying binary files and, because of its small size, can be used with larger programs than UPD2. The component name for UPD1 is "CHUP1?? UPD1 UTIL", but for the benefit of the user, the program is released under its common name: UPD1.

UPD1 runs in upper memory and overlays part of the monitor. This means that you cannot exit from the utility to the monitor directly. UPD1 requires the retrievable device drivers that reside on the system medium. The system medium must therefore remain online and ready throughout your use of UPD1. UPD1 can only use one device at a time. It cannot transfer files.

This chapter is organized as follows. First there is a brief description of how to start UPD1 and a list of commands (pg. 44). Next there are detailed descriptions of each command. And finally there is a list of error messages with a description of each. For a quick reference for error messages, look in the index under "error messages (UPD1)".

Starting UPD1

To start UPD1, type:

```
R UPD1
```

When the program has been successfully loaded by the monitor, it will type its name and restart address and then type a prompt, "*", to tell the user that it is ready to accept commands. The commands are tabulated below and described in detail in this chapter.

UPD1 Commands

CLR	clear UPD1 program buffer
LOAD	load a program
MOD	modify file image in memory
XFR	set transfer address
HICORE	set upper memory limit for dump
LOCORE	set lower memory limit for dump
DUMP	dump a program image
DEL	delete a file
BOOT	bootstrap a device

File Modification

The function of UPD1 is to modify binary files. When a deficiency is found in a diagnostic program, one of the corrective measures taken is to issue a DEPO or "patch order". This is a temporary change to a released program. UPD1 is one of the means of implementing these temporary remedies. The program in question is loaded from an XXDP+ medium into an area in memory called the program buffer. This area lies in the physical memory space between the UPD1 program and the XXDP+ monitor. The program image, now resident in memory, may be modified and then put back onto an XXDP+ medium ("dumped").

The next seven commands relate to this function of UPD1. In the descriptions of these commands, locations within a program that has been loaded into the program buffer are referred to as "virtual locations" since their addresses are relative to the first physical location in the program buffer and not the first physical memory location as would be the case if the program had been loaded by the monitor.

The location addresses given in a DEPO (Diagnostic Engineering Patch Order) are treated as virtual when using UPD1. For example, if the DEPO says to modify location 1002, that will be the virtual location you will refer to in the MOD command. If the program has been loaded by the monitor however, 1002 would be an absolute address in memory, not a relative location in the program buffer.

The file modification process applies to image (BIC or BIN) files only. The process, briefly, consists of the following steps:

1. Load the file into the program buffer with the LOAD command.
2. Change the size of the image, if necessary, with the HICORE and LOCORE commands.
3. Modify the contents of the desired location(s) with the MOD command.
4. Modify the transfer address, if desired, with the XFR command.
5. Write the image onto media using the DUMP command.

CLR

The clear command clears the program buffer into which programs are loaded for modification. This command allows the user to assure that unused locations in a program are set to zero when the program image is dumped to a medium. The format of the command is:

```
CLR
```

LOAD

The load command is used to load a binary file into memory for the purpose of modifying the program image. The command format is:

```
LOAD [devi:]ifile
```

where,
devi - device from which to load the file; default is the system device; device is assumed to be online and ready

ifile - file to be loaded; no default accepted; wildcards are not allowed

The program buffer is not cleared between loads and the unused locations for the last file loaded will not necessarily contain zero's for contents.

After the load has been successfully completed, UPD1 will print the transfer address and core limits (the lowest and highest virtual memory locations used by the program). These parameters may be altered by the user as described in subsequent command descriptions. Example:

```
LOAD DX0:PROG1.BIN

XFR: 000001    CORE: 000000,020000
```

MOD

The modify command is used to alter the contents of one or more virtual memory locations in a program that has been loaded by UPD1. The format of the command is:

```
MOD nnnnnn
```

,where nnnnnn is the octal address of the virtual memory location where the contents are to be modified. UPD1 will respond by typing the address and the current contents. It will then wait for the user to either type new contents or accept the the current contents by typing a ''carriage return''. If the user wishes to modify, or examine, two or more consecutive virtual memory locations, he or she should type a ''line feed'' after modifying each location. Examples:

```
MOD 2460
002460 770  771(CR)
```

The user has modified virtual memory location 2460.

```
MOD 12004
012004 012736(CR)
```

The user has examined, but t modified, virtual location 12004.

```
MOD 1220
001220 120  167(LF)
001222 120  1234(CR)
```

The user has modified two consecutive virtual memory locations (1220 and 1222).

XFR

This command is used to modify the transfer address in the program that has been loaded. The transfer address is the address at which a program will be started after being loaded into memory. The file created by the DUMP command will have this transfer address. This is address of the location to which control will be transferred when the program is started. The transfer address is always set to "1" when a program is loaded into the program buffer. (The transfer address in the file loaded from is not altered.) A transfer address of "1" is equivalent to a transfer address of "200".

After the user has entered this command, UPD1 will print the current transfer address. If the user does not wish to alter the transfer address, he or she should immediately type a "carriage return". The format of the command is:

```
XFR
```

Examples of this command:

```
XFR
000200 001000
```

The user has changed the transfer address from 200 (octal) to 1000 (octal).

```
XFR
002000(CR)
```

The user has not altered the transfer address.

HICORE

The high core command is used to alter the address of the highest virtual memory location that will be transferred during a "dump" operation. The default location is printed after the LOAD command. The format of this command is:

```
HICORE
```

The address of the highest virtual memory location to be used during a dump operation will be printed. UPD1 will then wait for the user to alter or accept the location. The user may enter a new location by typing the octal address or accept the location as is by typing a "carriage return". This address must be above that of the lowest virtual location (low core) and below that of the top of the program buffer.

Examples:

```
HICORE
40000 45000
```

The user has changed the upper virtual location from 40000 to 45000.

```
HICORE
100000 (CR)
```

The user has allowed the upper location to remain as 100000.

LOCORE

The low core command is used to alter the address of the lowest virtual memory location that will be transferred in a dump operation. The default location is printed after the load operation has been completed. The format of this command is:

```
LOCORE
```

Examples:

```
LOCORE
000200    0
```

The user has modified the low memory address to zero.

```
LOCORE
000000    20
```

The user has raised the low memory address to 20 (octal).

DUMP

The dump command is used to write the program image in the program buffer into a file on a medium. The image size is determined by the upper and lower memory limits displayed by the HICORE and LOCORE commands. A transfer address will be put into the file. This address can be examined and altered with the XFR command. The format of this command is:

DUMP [dev:]ofile

where,
dev - device to which file is to written; default is system device; device is assumed to online and write-enabled

ofile - file name for image file; wildcards are not accepted; file with specified name must not already exist on device

Examples of the dump command:

DUMP DY0:ZRLAA1.BIN

The program image will be written to DY0 and given the file name: 'ZRLAA1.BIN'.

DUMP MM0:ZXXAB2

The program image will be written to MM0 and given the name: 'ZXXAB2'. There will be no extension since one was not specified.

DUMP FILE3.BIC

The image will be written to the system device and given the specified name.

Sample Modification

The following sample file modification illustrates the use of the preceeding seven commands. It is based on the following patch for a fictitious diagnostic named ZXXXB0.BIN:

Location	Old Contents	New Contents
1224	106701	240
1226	177660	240
1452	376	374

The UPD1-user dialogue used to accomplish this is shown below. The underlined portion is that typed by UPD1. (LF) and (CR) refer to line feed and return.

```
.R UPD1

CHUP1B0 XXDP+ UPD1 UTILITY

RESTART ADDRESS:  002432

*LOAD ZXXXB0.BIN

XFR:  000001       CORE:  000200, 027230

*MOD 1224
001224    106701     240(LF)
001226    177660     240(CR)

*MOD 1224
001224    000240     (LF)
001226    000240     (CR)

*MOD 1452
001452    000376     374(CR)

*MOD 1452
001452    000374     (CR)

*DUMP ZXXXB1.BIN

*
```

The user examines locations previously modified in order to verify the changes. The user also renames the file to reflect the patch level.

DEL

The delete command is used to remove a file, or files, from anXXDP+ medium. The actual process for deletion is to remove the file name from the directory and deallocate the physical blocks used by the file. The format of the command is:

```
DEL dev:ifile
```

where,
- dev - device where file resides; device is assumed to be online and write-enabled; there is no default
- ifile - name of file to be deleted; no default accepted; no wildcards allowed

BOOT

The bootstrap command is used to start the monitor in the same manner as the hardware bootstrap. The purpose of this command is to allow the user to boot a device other than the original system device. The booted device is now the system device. The format of the command is:

BOOT dev:

The device must have a bootable medium mounted. The boot process consists of loading the first physical block (boot block) into the first 256 (decimal) words of memory and starting execution at location 0.

UPD1 Error Messages

? CKERR
Checksum error occured while loading a file.

? DEV FULL
The output device has no room for the last file specified.

? device error DIRECTORY
The specified device error occured while reading/writing to the directory. Possible device errors are: READ ERROR (error occured while reading a block of data), WRITE ERROR (error occured while writing a block of data) and HARD ERROR (error occured while accessing the device for a non-transfer operation).

? device error ON INPUT DEVICE
Specified error occurred on the input device specified in the last command.

? device error ON OUTPUT DEVICE
Specified error occurred on the output device specified in the last command.

? EOF
End-of-file indication was detected before expected.

? file already exists
The file name specified for output in the last command is already in the directory. Use another name or delete the current file.

? INV ADR
The address specified in the last MOD command was not a valid address in the program buffer.

? INV CMD
The previous command is not a recognized UPD1 command. Check the appendix for a list of valid commands and reenter the command.

? INV DEV

User specified an invalid device in the last command. An example would attempting to delete a file on the papertape reader.

? INV NAME

Filename specified in last command was not valid. Check command and re-enter.

? INV NUM
A number specified in the MOD command was not octal.

? NEXFIL
Specified file name for input in the previous command does not

appear in the directory.

? NO DEFAULT
The previous command does not accept default specifications for device and/or file name. Refer to the description of the command in question.

? OVERFLOW
The file specified in the previous LOAD command cannot fit in free memory space. If you are attempting to patch the file in question, use the PATCH utility.

Chapter 13 PATCH

This program can be used to modify any binary formatted (.BIN or .BIC) file stored on an XXDP+ storage medium. It is an alternative to the LOAD-MOD-DUMP sequence of UPD1 and UPD2.

There are two specific instances when this program should be used. The first is if you are modifying a file which is too large to be loaded into the memory space of the system you are using. This situation precludes the use of the LOAD-MOD-DUMP sequence of the update programs. The second use of this program is for the modification of DEC/X11 runtime exercisers. (It is assumed that reader is familiar with DEC/X11 usage. If not, please read the DEC/X11 User Manual before attempting to use PATCH on this software.) As these programs CANNOT be patched using the update programs, you must use this program if you wish to produce a permanently modified .BIN file for a DEC/X11 RTE.

****
Notice
The DEC/X11 features have
not been fully implemented in
DEC/X11 monitor. Please
patch RTE's with this
utility as you would any other
(non-DEC/X11) binary file.
****

Program Operation

Operation of this program consists of two phases. The first is the building of a table containing the modifications that will be made to the file in question. This table is referred to as the "input table". The operator fills this table with the addresses he wishes to modify within the file, along with the desired contents of these addresses. This table may then be saved as a file and retrieved for later use. The second phase of operation is the combining of the information contained in the input table with the actual binary file to produce a new, modified file. The original file is not modified by the program.

Running PATCH

In order to load and start this program, you must type the following command to the XXDP+ monitor:

.R PATCH <CR>

This will cause the PATCH utility to be loaded into memory and begin executing. The program will identify itself with the message:

CHUPA?? XXDP+ PATCH UTILITY

This will be followed by an operator prompt (*). At this point you may begin entering commands to the program. The valid commands for PATCH are:

BOOT	Boot specified device
CLEAR	Clear input table
EXIT	Return to XXDP+ monitor
GETM	Load DEC/X11 MAP file
GETP	Load saved input table
KILL	Delete address from input table
MOD	Enter address in input table
PATCH	Create patched file
SAVP	Save input table
TYPE	Print input table on terminal

In order to patch a file with this program you must perform two operations.

1. You must build an input table containing all of the addresses which you wish to modify within the file, along with the contents you want these addresses to have. The input table may have a maximum of fifty (50) of these entries.

   The commands you may use to build the input table are:

   CLEAR
   GETM
   MOD
   TYPE
   KILL
   SAVP
   GETP

   These commands are described below.

2. After you have completed the input table you must use the PATCH command to add the address modifications within the input table to the file you want to patch. The PATCH command is described below.

   It is important to note that the file you are modifying is never completely loaded into memory.

Patching Commands

CLEAR

This command clears the input table of all entries. The command format is:

    CLEAR<CR>

[illegible]

[illegible]mand is used [illegible] when patching DEC/X11 runtime exercisers. It will retrieve a DEC/X11 "MAP" file of the specified filename from the specified device and load it into memory (see section: "DEC/X11 MAP FILES"). The command format is:

    GETM [dev:]filnam.ext<CR>

The default device is the system device.

MOD

The MOD command is used to enter an address and the desired new contents of that address into the input table. The MOD command has two modes of operation, depending on whether you are modifying a DEC/X11 RTE or another type of binary file. These two modes are described in the following sections.

Non-DEC/X11 Mode

When used with binary files other than DEC/X11 RTE's, the format of the MOD command is:

    MOD <addr><CR>

where <addr> is any valid 16-bit address. Leading zeros may be omitted.

After a carriage return is typed the requested address will be retyped, followed by a slash. If this address has not been previously entered in the input table, the slash will be followed by six dashes. (Because the file you wish to modify is not in memory, there is no way of knowing the current contents of the location you have specified.) [illegible] the address has been inserted into the input table, the previously entered contents of the address will by typed after the slash.

Example:

```
*MOD 123456<CR>
123456/------
```

In this example, the operator has specified physical address 123456 to be modified. The dashes indicate that this is the first time this address has been specified.

Example:

```
*MOD 11040<CR>
011040/000240
```

In this case an actual value appears after the slash. This indicates that the operator had previously entered this address into the input table and had specified the new contents of the address to be 000240.

At this point you may type the value you wish to have loaded into this address. This value can be any octal number from 0 to 177777. Leading zero's may be omitted.After entering the value, you may type either a <CR> or a <LF>. A <CR> will close the table entry for this address and cause a prompt to be printed so that another command may be typed. A <LF> will close the current table entry and make a new entry for the next addressable memory location (i.e. <addr>+2). The new contents for this address may then be typed.

Example:

```
*MOD 123456<CR>
123456/------ 000207<CR>
*
```

In the example, the operator has specified that location 123456 should contain 000207. The carriage return closes the input table entry and causes a prompt (*) to be printed.

Example:

```
*MOD 11040<CR>
011040/000240 000137<LF>
011042/------ 051502<CR>
*
```

In this case the operator has re-opened the table entry for address 11040 and changed the contents to 000137. He/she then typed a <lf> to make a table entry for location 11042. The dashes indicate this location had not been previously entered into the table. This location receives a contents of 51502, then a <CR> is typed to close the table entry and cause a prompt to be printed.

DEC/X11 Mode

When working with DEC/X11 files, the command has three different formats, as follows:

```
MOD <addr><CR>
MOD MON <modnam> <addr><CR>
MOD <opmod> <addr><CR>
```

Form (1) of the MOD command is the same for DEC/X11 usage as it is for non-DEC/X11 usage (see previous section), with the exception that 18-bit addresses accepted.

Forms (2) and (3) of the MOD command allow the operator to specifiy locations within a DEC/X11 runtime exerciser (RTE) by typing the name of a monitor module or option module followed by an offset into that module. These forms of the command may be used only if a DEC/X11 ''map'' file has been retrieved by means of the GETM command (see section: ''DEC/X11 MAP FILES'').

Format (2) is used for modifying locations within monitor modules. An example of this format is:

```
MOD MON KTERR 24<CR>
```

The keyword ''MON'' is used to indicate that the module is in the monitor section of the RTE. In this case the operator is specifying location 24 relative to the beginning of the monitor module named ''KTERR''.

Format (3) is used for modifying locations within option modules of exercisers. <opmod> is the name of the option module to be modified. The name has five characters, the fifth character being the copy number (the first copy is 0). An example of this format is:

```
MOD CPBJ0 100<CR>
```

The operator, by typing this command, is indicating that he wishes to modify location 100 relative to the beginning of the first copy of option module CPBJ.

In all three formats, after the <CR> is typed the actual physical address being referenced will be typed, followed by a slash (/). If this address has not been previously entered in the input table, the slash will be followed by six dashes (------). If the address has already been entered in the input table, the previously entered contents of the address will be printed after the slash.

Example:

```
*MOD 012546<CR>
12546/------
```

In the example, the operator has specified physical address 12546 to be modified. The address is retyped, followed by a slash.

The dashes indicate that this is the first time this address has been specified for modification.

Example:

```
*MOD MON KTERR 10 <CR>
007126/000240
```

In this case the operator has specified an address within the monitor section of the RTE and has done so by using a module name. Notice that the program determines the actual physical address represented by the command string arguments and prints that address. Here an actual value appears after the slash. This indicates that the operator had previously entered this address into the input table and specified new contents of 000240.

At this point you may type the value you wish to have loaded into the specified address. This value can be any octal number from 0 to 177777. Leading zeros may be omitted. After entering the value you wish, you may type either a carriage return or a line feed. A <CR> will close the current table entry and cause a prompt (*) to be printed so that you can then type another command. A <LF> will close the current table entry and make a new entry for the next addressable memory word (i.e., <addr>+2>. You may then type new contents for the address.

Example:

```
*MOD 012546<CR>
12546/------ 000207<CR>
*
```

In the example, the operator has specified that location 12546 should contain 000207. The <CR> closes the input table entry and causes a prompt to be printed.

Example:

```
*MOD MON KTERR 10<CR>
007126/000240 137<LF>
007130/------ 51502<CR>
*
```

In this case the operator modified the contents of the location 7126 so it will be 000137 (he had previously specified that this address should contain 000240). He then typed a <LF> to close the table entry for address 7126 and to create a table entry for location 7130. The dashes indicate that this location had not been previously entered in the table. This location received a contents of 51502 and a <CR> was typed to close the table entry and cause of prompt to be typed.

TYPE

This command causes the contents of the input table to be listed on the system terminal.The command format is:

TYPE<CR>

KILL

The KILL command is used to delete an entry from the input table. The command format is:

KILL <addr><CR>

SAVP

This command causes the contents of the input table to be saved as a file with the specified filename on the specified device. The command does not cause any alteration to the input table contents. The command format is:

SAVP [dev:]filnam.ext<CR>

The default device is the system device.

GETP

The GETP command causes the input table to be loaded with the contents of a file that was created using the "SAVP" command. Execution of this command causes any previous contents of the input table to be lost. The command format is:

GETP [dev:]filnam.ext<CR>

The default device is the system device.

Creating the New File

PATCH

After the device address modifications have been entered in the input table, a new output file containing these modifications can be produced with the use of the PATCH command. This command reads the specified input file, adds the address modifications contained in the input table, and builds a new output file having the specified file name. The format of the command is:

PATCH [dev:]filnam.ext=[dev:]filnam.ext<CR>

The default device (input and output) is the system device. The input file and the input table are unaffected by the execution of this command. An example command string is:

PATCH DK1:SAMPL2.BIN=DK0:SAMPL1.BIN<CR>

This will take the file SAMPL1.BIN located on device DK0, combine it with the address modifications in the input table, and produce a new file on device DK1 called SAMPL2.BIN.

After the carriage return is typed, the following instruction will be printed:

IF THIS IS DECX11 TYPE THE MONITOR TYPE. ELSE JUST <CR>

If you are patching any file which is not a DEC/X11 runtime exerciser, then just type a <CR>. If this response is typed, the program will commence construction of the output file. When execution has been completed the message 'DONE'' will be printed, followed by a prompt.

If you are patching a DEC/X11 runtime exerciser (RTE) you must respond to the printed question by telling the program the type of monitor contained in the RTE. Note: There are three methods for determining the monitor type of the RTE:

1. Run the DEC/X11 configurator/linker program and type the configuration file for this RTE, if it exists.

2. Run the DEC/X11 configurator/linker program and type the MAP file for this RTE, if it exists. The monitor type will be at the top of the listing.

3. Run the RTE. The monitor type is printed at start-up time.

If the monitor of the RTE is one that does not support memory management, the program will now begin building the new output file. When this process is complete, the message 'DONE'' will be printed on the terminal, followed by a prompt. If, on the other hand, the specified DEC/X11 monitor type is one that does support memory management, the program will now check to see if a MAP file has been loaded into memory with the GETM command. If there is no MAP file in memory, the following instruction will be typed:

TYPE MODQ ADDRESS:

In order to determine this address, you must look at a listing of an appropriate MAP file (see DEC/X11 MAP Files). The symbol MODQ is located within the monitor module ''CONFIG'', so just find the module name ''CONFIG'' on the listing, then look at the symbol names underneath the module name until 'MODQ'' is found. The physical address printed next to the symbol 'MODQ'' is the address which must be typed in response to the question. If a MAP file was previously loaded into memory using the GETM command, the program can automatically find this address and thus will not ask the question.

Next, the following message will be printed:

```
IF MODIFYING OPTION MODULES, TYPE LOWEST MODULE
 ADDRESS, ELSE JUST <CR>
```

If you are modifying only the monitor section of the runtime exerciser, the proper response to this message is to simply type a <CR>. If you are patching both the monitor area and one or more option modules, or if you are patching only option modules, you must now type the address of the first option module that was linked into the runtime exerciser. This address may be obtained in two ways:

1. If you have a MAP file listing for the proper monitor type (see 'DEC/X11 MAP FILES''), find the first occurance of an option module name on the listing and use the physical address associated with that option module. The option modules are located at the end of the listing. The physical address of each module is printed next to the module name, under the heading 'PH ADDR''.

2. Before running the PATCH program, you can load and run the RTE that you intend to patch, then type a 'MAP'' command. This command causes the starting addresses of all option modules to be printed. For each option module, look for the address labelled 'PA:''. Find the physical address which is lowest (not necessarily the first one printed!).

After the typing of a <CR> the program will commence construction of the output file. When execution has been completed the message 'DONE'' will be printed, followed by a prompt.

Exitting

EXIT

This command returns control to the XXDP+ monitor. The command format is:

```
EXIT <CR>
```

BOOT

This command boots the specified device. The format is:

```
BOOT [dev:]<CR>
```

The default device is the system device.

DEC/X11 MAP Files

In order to utilize the full capabilities of the MOD command when modifying DEC/X11 runtime exercisers, you must have a MAP file. The MAP file is produced by the DEC/X11 configurator/linker. It is the symbol table which is generated at link time and saved using the SAVM command. (Please refer to the DEC/X11 User Manual.) Without the MAP file the MOD command will only accept physical addresses as arguments, but if a MAP file has been fetched using the GETM command, the MOD command will accept module names (both monitor and option module names) as arguments.

If you are going to modify the option modules of a particular runtime exerciser, and if you wish to be able to type the option module name and an offset value when using the command, then you must use the MAP file generated during the linking of that particular RTE. This is because the number and order of the option modules in any RTE is unique. If you don't have the proper MAP file you will have to manually calculate the physical address of the option module's relative address and type that value as the MOD command's argument.

On the other hand, for any given monitor type (A, B, C, etc.), the monitor modules are always linked in the same sequence. This implies that if you are modifying the monitor section of an RTE and wish to type the monitor module name plus an offset when using the MOD command, you need not have the MAP file for the particular RTE you are modifying. Any MAP file for the proper monitor type will do. For example, if you are modifying the monitor area of an RTE of a monitor type C, you may use any MAP file which was generated when linking any RTE having monitor type C. Similarly, the address of MODQ is also the same for every monitor of the same type, so if the PATCH command prompts you for the address of MODQ, you may look at any map listing of the proper monitor type to obtain the address.

It is important to remember that when a new release of the DEC/X11 monitor library is issued, all MAP files generated from the previous release of the library become invalid. They may not be used when patching files generated with the new library. New MAP files must be produced.

Suggested DEC/X11 Application

When a DEPO is issued for a DEC/X11 monitor module, the patch must be added to every runtime exerciser that is generated containing that module (depending on the monitor type). If you are in an environment in which you build many RTE's (manufacturing, for example), it is suggested that you build and save an input table for every monitor type. These saved tables can be added to and re-saved every time a new DEPO is issued. After any RTE is built with the configurator/linker, simply run this program, get the input table for the proper monitor type, add to the table any modifications that must be made to the option modules (using the MAP file if you wish) and then execute the PATCH command.

Error Messages

The following is a list of all error messages printed by this program:

? ADDRESS NOT FOUND

The specified address does not exist as an entry in the input table.

? CHECKSUM ERROR

A checksum error occurred while attempting to read the specified input file.

? COMMAND NEEDS ARGUMENT

The command typed by the operator requires an argument, but none was given.

? DELETE OLD FILE

The specified output filename already exists.

? device error WHILE LOADING DRIVER FOR xx
? device error WHILE READING
? device error ON INPUT DEVICE
? device error ON OUTPUT DEVICE
? device error DIRECTORY

The specified device error occured during the operation indicated. Possible device errors are: READ ERROR (error occurred while reading a block of data), WRITE ERROR (error occurred while writing a block of data) and HARD ERROR (error occured during a non-transfer operation).

? END-OF-MEDIUM

While reading a file, the end of the file was encountered before it was expected.

? FILE NOT FOUND

The specified input filename does not exist.

? FILE TOO BIG

? INPUT TABLE EMPTY

The specified command cannot be executed because there are no entries in the input table.

? INPUT TABLE FULL

The input table is full and cannot accept any more entries.

? INVALID ADDRESS

An address given in the last command was not legal (possibly an odd number). Check the command and re-enter properly.

? INVALID COMMAND

The last command entered was not a valid command for the PATCH Utility. Check the command (especially spelling) and re-enter properly.

? INVALID DEVICE

The specified command does not exist.

? INVALID FILE NAME

The specified filename does not have the correct format.

? INVALID MODULE NAME

A DEC/X11 module name was incorrectly specified.

? INVALID NUMBER

The specified number was not octal.

? MODULE NAMES NOT ALLOWED WITHOUT MAP FILE

The operator attempted to specify a module name in the MOD command without first loading the proper MAP file.

? MODULE NAME NOT FOUND

The specified module name does not exist within the DEC/X11 runtime exerciser.

? MUST BE EVEN

The operator attempted to specify an odd number as an address.

? MUST BE OCTAL

The operator attempted to type a non-octal number.

? NEED NUMBER

The operator omitted a numeric value from a command that expected one.

? NO DEVICE DEFAULTS

Default device names are not allowed.

? NOT ENOUGH ROOM TO LOAD DRIVER

The driver for the specified device will not fit into memory.

? NOT FOUND

? NUMBER TOO BIG

The value typed was too large for its intended purpose.

? OPTION MODULE NAME NOT FOUND

The specified option module does not exist.

? SPECIFY DEVICE

There is no default for device in the last command entered. Specify the device.

? SYNTAX ERROR

The last command was not formatted correctly. Refer to detailed descriptions in this chapter.

? UNEXPECTED END-OF-FILE

End-of-file indication was detected before all expected blocks were found.

?WRONG MAP FILE FOR MONITOR TYPE

The MAP file in memory does not have the specified monitor type.

Chapter 14 SETUP

It is assumed that the reader has read Chapters 8, 9 and 10 and is familiar with the Diagnostic Runtime Services (DRS).

SETUP is an XXDP+ utility that allows the user to build the hardware and software tables for a diagnostic prior to running the diagnostic and store the tables with the diagnostic. SETUP also combines a special version of DRS with a diagnostic for use with ACT and SLIDE. These are DEC manufacturing systems.

SETUP has the same memory requirements as DRS: 5.75K words. The minimum size system that can be used is 16K words.

To run SETUP, use the XXDP+ run command. The first thing that SETUP will do is ask for the type of environment you are going to be using, XXDP+ or ACT/SLIDE. Below is an example of starting SETUP for use in building XXDP+ environment diagnostics. The underlined portions are typed by the system.

```
.R SETUP
TARGET ENVIRONMENT:  XX(DP) OR AC(T)?  XX
*
```

SETUP is now ready to accept commands. There are only three commands in SETUP:

SETUP	build tables for specified diagnostic
LIST	type a list of DRS diagnostics on a medium
EXIT	return control to XXDP+

SETUP

The SETUP command will cause the specified diagnostic to be loaded into memory. SETUP will then process the table building code in the diagnostic. The user will go through the same process that would occur if actually running the diagnostic and issuing a START command. Section 4 of Chapter 9 describes the process in detail. The format of the command is:

```
SETUP [devo:]ofile=[devi:]ifile
```

where,

devo - device to which file is to be written; default is system device; device must be on-line

ofile - name of file for the diagnostic that has been SETUP

devi - device from which file is to be read; default is system device; device must be on-line ifile - name of file for the diagnostic that is to be SETUP

You may give the output file the same name as the input file, but you will get a warning message if you do. This is to avoid accidental loss of the original file. The message is:

```
DELETE ifile?  (Y/N/CR=Y)
```

If you type a ''Y'' or no answer at all, the input file ''ifile'' will be deleted after the SETUP process and the new file will then be written. If you specified the ACT environment, the file ''HSAC??.SYS'' will be appended to the output file. If you answer ''N'' to the question, SETUP returns to prompt mode so that you may enter a new command.

LIST

The LIST command is used to obtain a list of all DRS-compatible diagnostics on a medium. The format of the command is:

```
LIST [dev:][file.ext]
```

where,

dev - device to search for DRS-compatible files; default is the system device

file.ext - file(s) to search; extension must be BIN or BIC; wildcard specifications are accepted; default is ''*.BI?''

Chapter 15 XTECO

The XTECO (pronounced "ex-tee-co") utility is used to create and edit (modify) text files. Text files contain ASCII data representing valid text. Valid text consists of all printing characters, tab, carriage return, line feed and form feed. The prime text files in XXDP+ are batch control (chain) files. This utility, which is a simple editor, is a limited subset of TECO, a character editor supported by most of DEC's operating systems. The commands are few and simple, but adequate for the task.

Before describing the commands in detail, let's look at some basic XTECO concepts. To XTECO, a text file contains one long string of characters sort of like beads on a string. It processes the file one character at a time. The utility can only have a certain number characters in memory, so only a segment of the string can be worked with any given time. There are special characters (carriage return and line feed) that act as signals to XTECO to tell it when one line of text ends and another begins. This allows the editor to manipulate lines of text as well as characters. The editor keeps track of where it is on the string with a pointer. This pointer is manipulated by the various commands and is used by XTECO to locate where new text is to be placed and old text removed or modified. This pointer may be moved back and forth over the portion of the string in memory at the time, but cannot go backwards into the portion of the string already processed and placed into the new file. Thus the user should understand two basic concepts: text as a string of characters and a pointer to locate the editor on that string.

As previously mentioned, only a certain segment of a text file may reside in memory at any given time. XTECO processes files by reading a segment from the input file into memory, doing any edit functions on that segment (as directed by the user) and then writing the segment into the output file. This process continues until the entire input file has been acted upon. In the case of the "TECO" command (see next section) where the input and output file may have the same name and reside on the same medium, XTECO creates a temporary output file which replaces the input file after the edit process is complete.

Starting the Edit Process

The first thing a user must do after starting XTECO is to initiate the process of editing text. (XTECO is started by typing "R XTECO" in response to the XXDP+ monitor prompt.) There are three commands to do this.

```
TEXT - used to create a new file
TECO - used to modify a file
EDIT - used to modify a file
```

Each of these commands puts XTECO into what is called "edit mode". While in this mode, text information can be created, deleted or modified. There is one difference between the EDIT and TECO commands. The TECO command may be used with random-access devices (i.e., disks) only and the input file name is all that is required. The output file name will be the same as that of the input file. An output file is created during the edit process and is used to replace the input file after completion of the edit process. The EDIT command can be used with any type of device, but the input and output devices must be different (either different device types or different units of the same type). The formats of these commands are:

```
TEXT [devo:]ofile
TECO [devi:]ifile
EDIT [devo:]ofile=[devi:]ifile
```

where,
  devo - output device where the new file is to be stored
  ofile - name of file for output
  devi - input device from which old file is to read
  ifile - name of input file

After one of these three commands has been executed, XTECO will be in "edit mode" and will now issue double quotation mark ("") for a command prompt instead of an asterisk. There are thirteen commands that are used in edit mode. These are listed by type of function in the following table.

Edit Mode Commands

Pointer Location

L - move the pointer line by line
C - move the pointer character by character
J - move the pointer to the beginning of text
ZJ - move the pointer to the end of text

Search

S - search for specified string in text now in memory
N - search for specified string in remainder of text file

Modify/Display Text

T - type text
D - delete character(s)
K - delete line(s)
I - insert text
A - append text to that currently in memory

Terminating Edit Mode

EX - exit edit mode

All commands are terminated by two ''altmode'' or ''escape'' characters. The altmode or escape key is usually the left-uppermost key on DEC terminals. The character is echoed on the terminal as '$' and is shown as such in examples in this chapter.

The following sections explain each edit mode command in detail. Following these explanations, there is a sample edit session. In examples where it is important to display the position of the pointer, a caret, '^', is used to designate the position within the sample text. This is for illustrative purposes only! This character is not used for this purpose in actual operation.

Pointer Location

L Command

The L command is used to move the pointer on a line-by-line basis. A line of text is a string of characters between carriage return/line feed sequences. (This sequence is produced by typing the ''return'' key on your terminal.) The pointer may be moved either backward or forward any number of lines in the text currently stored in memory. The pointer is always positioned at the beginning of a line after execution of the L command. The format of the command is:

[n]L$$

where n is an optional argument that specifies the direction to move and the number of lines encompassed by the move. It is a decimal number that is positive for forward motion and negative for backward motion. If it is not specified, "1" is assumed. (The command is terminated by two altmode or escape characters which are echoed on the terminal by "$$" as shown above.)

Following are examples of the L command. In these examples, a caret, "^", indicates the position of the pointer after execution of each command. XTECO's prompt ("") is also shown in the examples.

```
Command        Text in Memory

 --            R ZRLA^??
               IF ERROR THEN
               PRINT RL01 HAS HARDWARE PROBLEM
               END

'L$$           R ZRLA??
               ^IF ERROR THEN
               PRINT RL01 HAS HARDWARE PROBLEM
               END

'-1L$$         ^R ZRLA??
               IF ERROR THEN
               PRINT RL01 HAS HARDWARE PROBLEM
               END

'2L$$          R ZRLA??
               IF ERROR THEN
               ^PRINT RL01 HAS HARDWARE PROBLEM
               END
```

C Command

The C command is used to move the pointer on a character-by-character basis. Lines are not recognized by this command; the carriage return/line feed sequence is treated as two characters. The pointer may be moved forward or backward any number of characters within the text currently stored in memory. The format of the command is:

[n]C$$

where n is an optional argument that specifies the direction to move and the number of characters encompassed by the move. It is a decimal number that is positive for forward motion and negative for backward motion. If it is not specified, "1" is assumed. (The command is terminated by two altmodes or escapes which are echoed on the terminal as "$$" as shown above.)

Following are examples of the C command. The caret, "^" indicates the position of the pointer after execution of each command. XTECO's prompt ("") is also shown in the examples.

```
Command          Text in Memory

 --             ;NEXT COMMAND WILL T^EST RP06
                ;ALL ERRORS WILL BE REPORTED

''C$$           ;NEXT COMMAND WILL TE^ST RP06
                ;ALL ERRORS WIL BE REPORTED

''-3C$$         ;NEXT COMMAND WILL^ TEST RP06
                ;ALL ERRORS WILL BE REPORTED

''10C$$         ;NEXT COMMAND WILL TEST RP06^
                ;ALL ERRORS WILL BE REPORTED

''3C$$          ;NEXT COMMAND WILL TEST RP06
                ;^ALL ERRORS WILL BE REPORTED
```

Please note the effect of the carriage return/line feed sequence on the execution of the last command in the example above. The sequence counts as TWO characters.

J Command

The J command is used to position the pointer at the beginning of all text currently stored in memory. The format of the command is:

J$$

The command is terminated by two altmodes or escapes which are echoed on the terminal as "$$" as shown above. Following is an example of the J command. The caret, "^", indicates the position of the pointer before and after command execution.

```
Initial state:
      R PROG1
      R PROG2
      R PROG^3

After execution of the J command:
      ^R PROG1
      R PROG2
      R PROG3
```

ZJ Command

The ZJ command is used to position the pointer after all text currently in memory. The format of the command is:

ZJ$$

The command is terminated by two altmodes or escapes which are echoed on the terminal as '$$' as shown above. Following is an example of the ZJ command. The caret, "^", indicates the position of the pointer before and after command execution.

```
Initial state:
      ^R PROG1
       R PROG2
       R PROG3

After execution of the ZJ command:
       R PROG1
       R PROG2
       R PROG3
       ^
```

Search

S Command

The S command causes the editor to search for a specified string of characters in the text currently stored in memory. Searches take place in the forward direction only. The search encompasses the text in memory only. The format of the command is:

Sstring$$

where "string" is the character sequence to search for. This string may consist of any number of valid characters, including tab, carriage return and line feed. The command is terminated by two altmodes or escapes which are echoed on the terminal as '$$' as shown above.

The editor will search through text in memory, starting from the current pointer position, until either a match is found or the end of text is encountered. An error message is printed if a match is not found. The pointer is positioned AFTER the string found by the search or after all text in memory if no match is made. (The user may reposition the pointer to the beginning of text in memory after a failed search by using the J command.)

Following are examples of the S command. The caret, "^", indicates the postion of the pointer after execution of each command. XTECO's prompt ('"') is also shown in the examples.

```
Command        Text in Memory

 --            ^R UPD2
               PIP DX0:=DX2:*.BIN
               EXIT

'SDX1$$        R UPD2
               PIP DX0:=DX2:*.BIN
               EXIT^

'J$$           ^R UPD2
               PIP DX0:=DX2:*.BIN
               EXIT

'SDX2$$        R UPD2
               PIP DX0:=DX2^:*.BIN
               EXIT
```

In the above example, the first search failed. The editor would print an error message:

```
'NOT FOUND:  DX1
```

N Command

The N command has the effect of a "non-stop" S command. It is exactly like the S command, except that it will search through all remaining text in a file. If the editor fails to find a match for the specified string within the text currently in memory, it will write the text in memory into the output file and bring more text in from the input file. This process continues until either a match is found or the entire input file has been checked. The J command may not be used to recover from a failed search if that search caused the section of text that you started from to be written to the output file. In this case, the user must exit edit mode and re-enter with the previous output file as input.

Modify/Display Text

T Command

The T command is used to type text on the console terminal. The text typed is relative to the position of the pointer in the text currently stored in memory. Typing is line-by-line and any number of lines before or after the current pointer position may be typed. A line of text is a string of characters between carriage return/line feed sequences. (This sequence is produced by typing the "return" key on your terminal.) If the pointer is positioned within a line of text, typing will start/conclude at the pointer position (see examples). The format of the command is:

[n]T$$

where n is an optional argument that specifies the number of lines to be typed and whether the lines preceed or follow the current pointer position. It is a decimal number that is positive if the lines follow the pointer and negative if the preceed it. If n is not specified, "1" is assumed. The command is terminated by two altmodes or escapes which are echoed on the terminal as "$$" as shown above.

There is also a special form of the T command which will cause all text currently stored in memory to be typed, regardless of the position of the pointer. This command is: "HT".

Following are examples of the T command. The caret, "^", in the sample text indicates the position of the pointer. XTECO's prompt ("'") is also shown in the examples.

First sample text:

```
     ;BATCH CONTROL FILE FOR TESTING THE DZ11
     ^R ZDZA??
     R ZDZB??
     ;END OF DZ11 TESTING
```

```
  Command      Text Typed
  'T$$       R ZDZA??
  '-1T$$     ;BATCH CONTROL FILE FOR TESTING THE DZ11
  '2T$$      R ZDZA??
             R ZDZB??
```

Second sample text:

```
     R PROG1
     R PRO^G2
     R PROG3
```

```
  Command          Text Typed
  'T$$           G2
  '-1T$$         R PROG1
                 R PRO
  '0T$$          R PRO
  'HT$$          R PROG1
                 R PROG2
                 R PROG3
```

D Command

The D command is used to delete characters from the text in memory. Any number of characters, either preceeding or following the current pointer position, may be deleted. The format of the command is:

[n]D$$

where n is an optional argument that specifies the number of characters to be deleted and whether the characters preceed or follow the current pointer position. It is a decimal number that is positive if the characters follow the current pointer position and is negative if they preceed it. If it is not specified, "1" is assumed. The command is terminated by two altmodes or escapes which are echoed on the terminal as '$$' as shown above.

Following are examples of the D command. The caret, "^", indicates the position of the pointer. XTECO's prompt ("'") is also shown in the examples.

```
Command        Text in Memory

 --         ;COM^MENT LINE IN BATCH CONTROL FILE

'4D$$       ;COM^ LINE IN BATCH CONTROL FILE

'-3D$$      ;^ LINE IN BATCH CONTROL FILE

'D$$        ;^LINE IN BATCH CONTROL FILE
```

K Command

The K command is used to delete lines of text from the text stored in memory. A line of text is a string of characters between carriage return/line feed sequences. (This sequence is produced by typing the "return" key on your terminal.) Deletion of a line includes the deletion of the terminating carriage return/line feed sequence in addition to the characters in the line. Any number of lines may be deleted either preceeding or following the current position of the pointer. If the pointer is positioned within a line, not all of the line will be deleted (see examples that follow). The format of the command is:

[n]K$$

where n is an optional argument that specifies the number of lines to be deleted and whether they proceed or follow the current pointer position. It is decimal number and is positive if the lines preceed the pointer and negative if the follow it. If this argument is not specified, "1" is assumed. The command is terminated by two altmodes or escapes which are echoed on the terminal as '$$' as shown above.

Following are examples of the K command. The caret, "^", indicates the position of the pointer after the execution of each command. XTECO's prompt ("'") is also shown in the examples.

```
Command        Text in Memory

 --            ;START OF CONTROL FILE
               ^R PROG1
               R PROG2
               R PROG3
               R PROG4
               ;END OF FILE

'K$$           ;START OF CONTROL FILE
               ^R PROG2
               R PROG3
               R PROG4
               ;END OF FILE

'-1K$$         ^R PROG2
               R PROG3
               R PROG4
               ;END OF FILE

'2K$$          ^R PROG4
               ;END OF FILE

'3C$$          R P^ROG4
               ;END OF FILE

'K$$           R P^;END OF FILE
```

Note the effect of the K command when the pointer is not positioned at the beginning of a line. You can easily determine the effect of any K command by issuing a T command with the identical format. Whatever is typed after you issue the T command is what will be deleted by the K command. (The commands 3T and 3K are of identical format.)

I Command

The I command is used to insert new text. The text is inserted after the current pointer position. The pointer will be positioned after the new text upon completion of the insertion. The format of the command is:

Itext$$

where "text" is the text to be inserted. This text may consist of any valid text characters. Valid text characters are all printing characters, tab, carriage return, line feed and form feed. The command is terminated by two altmodes or escapes which are echoed on the terminal as ''$$'' as shown above.

Following are examples of the I command. The caret, ''^'', indicates the position of the pointer after execution of each command. For purposes of illustration, line terminators are depicted in these examples. The return typed by the user is represented as ''<RET>''. The two character sequence generated by the return and stored in the text is represented as ''<CR><LF>''. XTECO's prompt ('''') is also shown in the examples.

Command	Text in Memory
--	R UPD2<CR><LF> ^
''IPIP$$	R UPD2<CR><LF> PIP^
''I DK0:=DK1:<RET> $$	R UPD2<CR><LF> PIP DK0:=DK1:<CR><LF> ^

Command	Text in Memory
--	FILE DK1:=DK^:<CR><LF>
''I2$$	FILE DK1:=DK2^:<CR><LF>

A Command

The A command is used to increase the amount of text stored in memory. This is done by reading the next section of text from the input file and ''appending'' it to the text already in core. You may append as many sections of text as memory size limits allow. The format of the command is:

A$$

The combined sections of text in memory are treated as a single section by the previously described commands.

Terminating Edit Mode

EX Command

The EX command is used when all editting operations have been completed. The output file is closed. If the edit session was initiated using the TECO command, the input file is renamed with a .BAK extension and the output file will be given the original name of the input file. XTECO will no longer be in edit mode as signified by the switching of the prompt character back to an asterisk ("*") from a double quote ('"'). The format of the command is:

```
EX$$
```

Combining Edit Commands

The user can combine several edit mode commands on a single command line. This is done by separating each command by a single escape (altmode) character and then terminating the entire string of commands by two escapes. For example, the following commands:

```
"NTEST$$
"0T$$
"T$$
```

can be combined into a single string:

```
"NTEST$0T$T$$
```

In both cases, XTECO will do a non-stop search for the string "TEST", type the characters from the beginning of the line where the string was found to the current pointer position and then type the characters from the current pointer position to the end of the line. Combining commands is merely a convenience for the user. It is suggested that the user not attempt to combine commands until he or she is familiar with the operation of individual commands. The next section has sample edit sessions that show both methods.

Sample XTECO Edit Session

What follows is a series of sample edit sessions that will show the user the various ways of handling XTECO. In these examples, the underlined text is that which is typed by XTECO. At appropriate locations, comments have been included for reader assistance. These comments are enclosed in square brackets ([]) and should not be confused with the actual dialogue that is taking place between user and software.

Simple Method for Creating a Text File

```
.R XTECO

*TEXT TEST.CCC
                    [User creating new file on
                         system device called TEST.CCC]
*I;THIS IS A BATCH CONTROL JOB FOR TESTING THE RX01
;THIS IS A FICTIONAL JOB FOR DEMO OF XTECO ONLY!
;
R ZRXX??
RES/PAS:1
N
EXIT
;THE FIRST RX01 DIAGNOSTIC HAS BEEN RUN.
;
R ZRXY??
RES/PAS:1/TES:1-5
N
EXIT
;END OF RX01 TEST
$$
*EX$$
*EXIT
.
```

Changing an Existing Text File

```
.R XTECO

*TECO TEST.CCC
          [The user is going to change TEST.CCC,
               on the system device.]
*SRX02$$
? NOT FOUND: RX02
*J$$
*T$$
;THIS IS A BATCH CONTROL JOB FOR TESTING THE RX01
*L$$
*2T$$
;THIS IS A FICTIONAL JOB FOR DEMO OF XTECO ONLY!
;
*EX$$
*EXIT
```

Use of Combined Edit Commands

```
.R XTECO

*EDIT DL1:TEST1.CCC=TEST.CCC

                     [EDIT TEST.CCC, WHICH IS ON THE SYSTEM
                      DEVICE, AND PLACE THE EDITTED OUTPUT
                      INTO A FILE CALLED TEST1.CCC ON DL1.]

'NPAS:$0TT$$         [SEARCH FOR 'PAS:' AND TYPE LINE]
RES/PAS:1
'DI2$0TT$$         [DELETE NEXT CHAR AND INSERT '2']
RES/PAS:2
'EX$$
*EXIT
.
```

Non-edit Commands

There are three XTECO commands that are not related to actual text file editting. These commands are provided for user convienience.

TYPE and PRINT

The TYPE and PRINT commands are used to print text files on the console terminal and line printer respectively. They are equivalent to the UPD2 commands of the same name.

EXIT

The EXIT command is used to return control to the XXDP+ monitor.

PART V -- XXDP+ Device Drivers

Chapter 16 Devices Supported

XXDP+ supports most mass storage devices. In addition it supports some non-file-structured devices such as paper tape. The following table lists all devices supported, the mnemonic used to specify the device and the name of the monitor and driver files. The "??" characters in the file name refer to the revision and patch level which may vary over time.

Device	Mnemonic	Monitor	Driver
TU60	CT	HMCT??	HDCT??
RP04/5/6	DB	HMDB??	HDDB??
TU58	DD	HMDD??	HDDD??
RK05	DK	HMDK??	HDDK??
RL01/2	DL	HMDL??	HDDL??
RK06/7	DM	HMDM??	HDDM??
RP02/3	DP	HMDP??	HDDP??
RM02/3	DR	HMDR??	HDDR??
RS03/4	DS	HMDS??	HDDS??
DECTAPE	DT	HMDT??	HDDT??
RX01	DX	HMDX??	HDDX??
RX02	DY	HMDY??	HDDY??
LOW SPD PT	KB	---	HDKB??
PRINTER	LP	---	HDLP??
TM02	MM	HMMM??	HDMM??
TS04	MS	HMMS??	HDMS??
TE10	MT	HMMT??	HDMT??
PDT11	PD	HMPD??	HDPD??
HI SPD PT RD	PP	---	HDPP??
LOW SPD PT	PT	---	HDPT??
HI SPD PT RD	PR	---	HDPR??

All drivers assume that the CSR address for the device is the standard address as given in the Peripheral Handbook. If you have a system with a device at a non-standard address, you can modify location 24 in the driver using UPD2 or UPD1. (The CSR address is stored in location 20 of the monitor.)

Chapter 17 Driver Error Messages

XXDP+ device drivers are, by necessity, small and limited in function. They can detect and report three types of errors: read, write and hard. These errors are reported and control is returned to the utility being used. The utility then takes any further action required. Since the functionality of the drivers is limited, the user is required to run diagnostics on the device in question if an error persists.

PART VI -- Building XXDP+

Chapter 18 Monitor and Required Files

The minimum files that must be put on a bootable XXDP+ medium are the monitor for that medium, the device driver for that medium, the DRS (file name: HSAA??.SYS) and the directory utility (file name: HUDI??.SYS). The monitor file (see Chapter 16 for names) must be loaded by UPD2 and then saved on the medium using either the SAVM or SAVE commands. These commands are described in Chapter 11. The SAVM command is used for random access (disk-type) devices. The SAVE command is used for sequential access (tape-type) devices. The remaining files may be put onto the medium using any UPD2 file transfer commands.

Examples (RX01 and TE10):

```
.R UPD2
*LOAD HMDX??.SYS
*SAVM DX0:
*FILE DX0:=HDDX??.SYS
*FILE DX0:=HUDI??.SYS
*FILE DX0:=HSAA??.SYS
*EXIT

.R UPD2
*LOAD HMMT??.SYS
*SAVE MT0:HMMT.SAV
*PIP MT0:=HDMT??.SYS
*PIP MT0:=HUDI??.SYS
*PIP MT0:=HSAA??.SYS
*EXIT
```

The process described above places the MINIMUM XXDP+ system on a medium. You may add as many other system components (drivers and utilities) as you wish. Don't forget to modify location 1000 in the monitor if you will be using a system that is on 50Hz power. (Location 1000 must contain a 0 for 60Hz and a 1 for 50Hz.) The next chapter describes a batch control file that builds media automatically.

Chapter 19 Update Kits

There is a batch control file that will update/build XXDP+ media automatically. The file is called XXBLD.CCC. To start the file, use the chain command. The file accepts switches that specify the media type to build and the mode in which to build. All supported XXDP+ media may be built. The media being built are always assummed to be mounted in drive (unit) 0 of the device and that the drive is ready and write-enabled. The format of the command line for starting the build process is:

```
C XXBLD/device[/mode]
```

where,

device - the mnemonic for the device to be built/updated. Supported devices and their mnemonics are listed in chapter 16. If no device is specified, a short help message will be printed.

mode - manner in which to build/update. Available modes are:

DRIVER	a bootable medium with all XXDP+ drivers
MONITOR	a bootable medium with all XXDP+ monitors
UTILITY	a bootable medium with all XXDP+ utilities
SYSTEM	a combination of the above three modes

If no mode is specified, a bootable medium is built. A bootable medium consists of a bootable monitor image, the runtime services, the directory utility, the driver for the medium and UPD2.

Except in the case of sequential devices (e.g.; magtape), the medium is not changed except for the replacement/addition of the new XXDP+ components specified by the mode switch. You may want to back up files that are critical however. Sequential media are destroyed. There is a warning message given and the user is given the opportunity to abort the process.

To obtain help while running the update batch job, use the following command:

```
C XXBLD/HELP
```

PART VII -- Batch Control (Chaining)

Chapter 20 Introduction to Batch Control

XXDP+ has a facility for running programs without operator intervention called batch control or chaining. The commands that would normally be issued by an operator are put into a text file (using XTECO - Chapter 15) and the monitor processes the commands in this file rather than requiring an operator to enter each command manually. Once a batch control file has been created, it can be used over and over again. The batch control process releases the operator from having to do repetitive tasks such as building new media or running a common set of diagnostics. More importantly, batch control allows a user to develop a test strategy and use the strategy consistently. This is done by selecting the proper diagnostics and running them in a particular order and mode to achieve the best test process. Once the process is developed, it is put into a batch control file.

Older versions of XXDP and XXDP+ had very limited batch control services. Essentially the user could "chain" together a series of run commands which would run various diagnostic programs, such as:

```
R PROG1
R PROG2
R PROG3
```

The user could intermix comments that would be printed as the chain was processed. This primitive process was adequate for most simple procedures, but was not adequate for more sophisticated operations such as the update kits described in Chapter 19.

XXDP+ now contains a fairly sophisticated set of batch control functions listed below. These are described in this chapter. Techniques for using these functions to run diagnostics and utilities are described in subsequent chapters.

Batch Control Functions

Monitor Commands	monitor commands: R, L, S, C, and E)
Utility Commands	UPD2, SETUP, etc.
DRS Commands	all DRS commands and diagnostic dialogue

conditionals	sections of the batch file can be processed conditionally under operator control or runtime conditions
GOTO tag	begin processing at another section of the batch file designated by "tag"
QUIET	inhibit printing of batch file if printing or enable printing if printing was inhibited previously
PRINT	temporary override of QUIET
SMI/CMI	enable/disable manual intervention operations in specialized diagnostics
QUIT	terminate the batch operation
WAIT	stop batch operation until the operator types a Control X

Monitor Commands

Certain of the monitor commands described in Chapter 6 can be used in a batch control file. They are the R, L, S, C and E commands. There are two functions which are different when used under batch control instead of operator control. First, the R (Run) Command has a pass switch for use with diagnostic programs which are not DRS compatible. The diagnostic may be run a certain number of passes by using the switch as shown:

```
R DIAG/5
```

DIAG will run 5 passes before the batch operation continues on.

The C (Chain Command) may be used in a batch file with one restriction. Batch operations can be nested one level only. That is, a batch file may start another batch file and then continue after the second file has been processed, but the second batch file may not start another (third) file.

With these exceptions, the monitor commands function under batch control as they would under operator control.

Utility Commands

The commands for various XXDP+ utilities may be used in batch control operations. Chapter 22 details this function.

DRS Commands

All of the DRS commands described in Chapter 9, including all switches and flags, can be used in a batch control file. All dialogue that would normally take place between an operator and a DRS diagnostic can also be placed in a batch control file. Chapter 21 describes this function.

Conditionals

Sections of a batch control file can be processed, or not processed, based on either operator input or certain conditions. There are three conditional statements.

```
1.  IF condition THEN
    statement(s)
    END

2.  IFERR THEN
    statement(s)
    END

3.  IFLMD n THEN
    statement(s)
    END
```

If the condition specified is true, the statements between the THEN and END statements will be processed. If the condition is false, these statements will be ignored.

The conditions used in the first type of statement are ASCII character strings which are defined by the person writing the batch file and used as switches by the operator. For example, suppose a person is writing a batch file for running UPD2 and doing some file operations. If a part of the process requires the presence of an RX02 on the system, there would be a need for the operator using the batch file to be able to specify whether or not there was an RX02 present. The batch file writer would define a conditional section of the file as shown below.

```
IF RX02 THEN
statement(s)
END
```

The condition ''RX02'' is now used by the operator as a switch to the Chain command:

```
C FILE/RX02
```

The monitor stores the string of characters for comparison with the conditions in the batch file. There is only one pre-defined switch and the writer is free to create any other he or she desires. The pre-defined switch is ''/QV'' (quick verify) which causes diagnostics to be run one pass only. Any number of switches may be used in a command.

The second type of conditional statement can be used with DRS-type diagnostics only. If a test error was detected by the last DRS-type diagnostic that was run in the batch file,the statements will be processed.

The third type of conditional uses the media-type byte in physical location 41. If the type code matches the one specified in the conditional ("n"), the statements will be processed.

GOTO

The GOTO statement is used to transfer control within a batch control file. When the monitor encounters a GOTO statement, it searches for the specified tag and resumes the batch process at the statement following the tag. A tag is an alphanumeric string terminated by a colon (":"). The tag may occur before or after the GOTO statement in the batch file. The following are examples of the GOTO steatement:

```
TAG1:
R PROG1
GOTO TAG1


GOTO TAG2
     .
     .
     .
TAG2:
R PROG5
```

In the first example, the batch process will loop backwards until interrupted by the operator. In the second example, control will be transferred forward to TAG2. Any statements between the GOTO statement and the tag are ignored.

QUIET

The QUIET statement is used to control typing of the batch file. The statement is used like a ""flip-flop". The first time the statement is encountered, all typing is suppressed (with the exception of error messages). The next time it is encountered, typing is reenabled. The third time it is encountered, typing is inhibited again and so on.

PRINT

The PRINT statement is used to force the typing of a line of text while typing is inhibited by the QUIET statement. The format of the statement is:

PRINT text

The text on the same line as the PRINT will be typed.

SMI/CMI

The SMI and CMI statements are used to enable and disable manual intervention modes in DRS-type diagnostics. Normally all testing that requires manual intervention by an operator are inhibited during batch control operations. These statements allow this to be over-ridden. Obviously caution is suggested when using this feature.

```
SMI - set (allow) manual intervention
CMI - clear (don't allow) manual intervention
```

CMI is the default state when a batch job is started.

QUIT

The batch job is immediately stopped when a QUIT statement is encountered. The monitor returns to normal operator mode.

WAIT

When a WAIT statement is encountered, the monitor stops processing the batch file and waits for the operator to type a CTRL-X (typed by depressing the CTRL and X keys together). This is typically used in conjunction with manual intervention feature as shown in the example below.

```
PRINT THE NEXT DIAGNOSTIC REQUIRES THAT A
PRINT SCRATCH MEDIUM BE MOUNTED IN THE RL01/02.
PRINT TYPE ^X WHEN READY
WAIT
R ZRLA??
```

Comments

Comments may be placed in the batch file. These will be typed as the file is processed unless QUIET mode has been invoked. Comments are strings of text that start with a semicolon.

```
;THE NEXT PROGRAM TESTS THE DZ11
.
R ZDZB??      ;RUN THE DIAGNOSTIC
```

Chapter 21 Batch Control of Diagnostics

For the purposes of batch control, there are two types of diagnostics: chainable non-DRS-type diagnostics and DRS-type diagnostics. The first type can be batched by a simple run command:

```
R DIAG[/n]
```

where n is an optional argument that specifies the number of passes that the diagnostic will run. The default is one pass.

S-type diagnostics require complete batch control. All mands normally entered by an operator must be in the batch ile. For example:

```
R DIAG2
START/PASS:1
Y       [answer for CHANGE HW]
1       [answer for number of units]
[insert answers for all HW questions]
EXIT    [to return control to batch job]
```

This is just a short example. The concept to note is that the batch file is an INDIRECT COMMAND file for DRS. All commands that are required when running under operator control are necessary in the file. If the diagnostic program in the above example had used a software table, it would have been necessary to provide the commands required to support it.

The user does not have to enter all commands via the batch file however. By using the SETUP utility (Chapter 14), all hardware and software information could be supplied to the diagnostic prior to running the batch job. This is the recommended method for using DRS-type diagnostics in the batch control environment. If you preset all the parameters, the following commands are all that are necessary:

```
R DIAG2
START/PASS:n
N
N
EXIT
```

where n is the number of passes to execute.

Chapter 22 Batch Control of Utilities

Most of the XXDP+ utilities may be used under batch control. The utilities which are batch controlable are: UPD2, SETUP, and PATCH. To run a utility under batch control, simply create a batch file that contains all of the commands that would normally be entered by an operator. For example, to build an RX01 floppy diskette for XXDP+ using UPD2 under batch control:

```
R UPD2
LOAD HMDX??.SYS
SAVM DX0:
FILE DX0:=HSAA??.SYS
FILE DX0:=HUDI??.SYS
FILE DX0:=HDDX??.SYS
EXIT
```

Note that the dialogue with UPD2 must end with an EXIT command in order to finish the batch job or to allow further batch functions.

PART VIII -- Appendices

Appendix A XXDP+ Secondary Bootstraps

To be supplied in future revisions.

Appendix B Bootstraps

To be supplied in future revisions.

Appendix C — Command Summary

Monitor Commands

R	run a program
L	load a program
S	start a program
C	run a batch job (chain)
D	list directory of load medium
F	set the terminal fill count
E	enable alternate system device
H	type help information
TEST	run batch file: SYSTEM.CCC

DRS Commands

Execution

START	start the diagnostic and initialize
RESTART	start diagnostic and do not initialize
CONTINUE	continue diagnostic at test that was interrupted by a ^C
PROCEED	continue from an error halt

Units Under Test

ADD	activate a unit for testing
DROP	deactivate a unit
DISPLAY	print a list of device information

Flags

FLAGS	print status of all flags
ZFLAGS	reset all flags

Statistics

PRINT	print statistical information

Exitting

EXIT	return to XXDP+ runtime monitor

DRS Command Switches

/TESTS:test-list	execute only the tests specified
/PASS:ddddd	execute ddddd passes (ddddd = 1 to 64000)
/FLAGS:flag-list	set specified flags
/EOP:ddddd	report end-of-pass after each ddddd passes (ddddd = 1 to 64000)
/UNITS:unit-list	command will affect only specified units

DRS Flags

Flag	Effect
HOE	halt on error - control is returned to runtime services command mode
LOE	loop on error
IER	inhibit all error reports
IBE	inhibit all error reports except first level (first level contains error type, number, PC, test and unit)
IXE	inhibit extended error reports (those called by PRINTX macro's)
PRI	direct messages to line printer
PNT	print test number as test executes
BOE	'bell' on error
UAM	unattended mode (no manual intervention)
ISR	inhibit statistical reports (does not apply to diagnostics which do not support statistical reporting)
IDR	inhibit program dropping of units
ADR	execute autodrop code
LOT	loop on test
EVL	execute evaluation (on diagnostics which have evaluation support)

UPD1 Commands

CLR	clear UPD1 program buffer
LOAD	load a program
MOD	modify file image in memory
XFR	set transfer address
HICORE	set upper memory limit for dump
LOCORE	set lower memory limit for dump
DUMP	dump a program image
DEL	delete a file
BOOT	bootstrap a device

UPD2 Commands

File Manipulation

DIR	give directory of specified medium
PIP	transfer a file or files
FILE	transfer a file or files
DEL	delete a file or files
REN	rename a file

File Modification

CLR	clear UPD2 program buffer
LOAD	load a program
MOD	modify file image in memory
XFR	set transfer address
HICORE	set upper memory limit for dump
LOCORE	set lower memory limit for dump
DUMP	dump a program image

New Medium Creation

ZERO	initialize a medium
SAVM	save a monitor on a disk
SAVE	save a monitor on a tape
COPY	copy entire medium

Miscellaneous

ASG	assign a logical name to a device
DO	execute an indirect command file
READ	read a file to check validity
EOT	write logical end-of-tape mark on a tape
DRIVER	load a device driver

Returning to Monitor

BOOT	bootstrap a device
EXIT	return control to the runtime monitor

Printing

PRINT	print a file on the line printer
TYPE	type a file on the console terminal

PATCH Commands

BOOT	Boot specified device
CLEAR	Clear input table
EXIT	Return to XXDP+ monitor
GETM	Load DEC/X11 MAP file
GETP	Load saved input table
KILL	Delete address from input table

MOD	Enter address in input table
PATCH	Crreate patched file
SAVP	Save input table
TYPE	Print input table on terminal

XTECO Non-edit Commands

TEXT - create new text file
TECO - modify a file on disk
EDIT - modify a file
TYPE - type a file on the console terminal
PRINT - print a file on the line printer
EXIT - return to monitor

XTECO Edit Commands

Pointer Location

L - move the pointer line by line
C - move the pointer character by character
J - move the pointer to the beginning of text
ZJ - move the pointer to the end of text

Search

S - search for specified string in text now in memory
N - search for specified string in remainder of text file

Modify/Display Text

T - type text
D - delete character(s)
K - delete line(s)
I - insert text
A - append text to that currently in memory

Terminating Edit Mode

EX - exit edit mode

Appendix D

Error Message Summary

Appendix E

Batch Control Summary

Appendix F User Tips

DRS Table Building

To save time and energy, prebuild the hardware and software tables in a diagnostic using the SETUP utility. Customize the files for a specific system on the XXDP+ medium for that system or customize files for several systems on medium shared between systems. Remember, you can always change the tables on the fly by using the START command or permanently change the files by using SETUP.

Another way to make XXDP+ work for you is to use the batch control functions described in Part VII of this manual. Familiarize yourself with the DRS-type diagnostics for a particular device and identify the various operating modes (as defined by the software tables) that are most useful for you. Prebuild the hardware tables for the system, or systems, you are working with. Then write a batch control file that implements the various modes based on conditionals. This allows you to enter one command to XXDP+ and then let the system do the rest.

A simple example of this type of batch control file is follows. For the purposes of this example, we will use a fictional diagnostic called "DIAG1". The normal operator dialogue with this diagnostic (with hardware tables already built) is:

```
.R DIAG1
DR>STA
CHANGE HW (L) ?  N
CHANGE SW (L) ?  Y
TEST ALL SECTORS (L) ?
```

The batch file that has been created for this diagnostic is called "DISK.CCC" and is listed below.

```
IF QV THEN
R DIAG1
STA/PAS:1
N
Y
N
END
IF REPAIR THEN
R DIAG1
STA/FLA:LOE
N
Y
Y
N
END
```

This file defines two test modes: quick verify and repair. Note how the batch file manipulates the software questions and also avoids answering the hardware questions since the tables were already created. The user invokes either of the two modes with one of the following commands:

C DISK/QV

or,

C DISK/REPAIR

Appendix G Glossary

This is a glossary of common terms used in connection with XXDP+.

autodelete - a possible effect of the file transfer process whereby a file from the input medium replaces a file of the same name on the output medium. In UPD2, only transfers initiated by a FILE command can result in autodeletion.

boot block - the first physical block an a medium (block zero). This block contains the XXDP+ secondary bootstrap for the device.

bootstrap - very simple code used to load and start more complex code from a medium such as a disk. The term comes from the phrase ''Picking oneself up by the bootstraps''. See also ''primary bootstrap'' and ''secondary bootstrap''.

buffer - a section of memory reserved for storing data, usually from a file, as opposed to executable code

console terminal - the video or hardcopy terminal attached to the system via the DL interface at bus address 177560.

device driver - that software which has the function of controlling the operation of a specific hardware component in a system. An RX01 driver, for example, is that software that accomplishes such tasks as selecting a physical block, reading a block of information, etc. on an RX01 disk .

device handler - see ''device driver''

dump - the process whereby an image of the contents of memory is placed on a storage medium.

edit - to modify text information in a file

editor - a utility program used to modify text files

hardware table - data structure where DRS stores the information regarding units being tested

load - the process whereby the contents of a file containing a program image are placed in memory.

medium - physical storage such as a disk or magtape. In this manual, the term ''medium'' is equivalent to ''XXDP+ medium''.

pass - a unit of diagnostic operation. A DRS-type diagnostic pass is defined to be execution of all specified tests on all active units.

patch - a temporary remedy for a problem in a program that is accomplished by altering the program image stored on the XXDP+ medium. See Chapters 11, 12 and 13 for descriptions of the various methods of altering the program image.

physical block - a group of data consisting of 256 (decimal) words. This is the standard size of data transmission to and from the XXDP+ media.

physical location - an absolute memory reference (see "virtual location").

primary bootstrap - code, usually stored in a ROM, which loads the "boot block" (block 0) from a medium into the first 256 (decimal) words of memory and then transfers control to memory location 0.

program buffer - a section of memory used by UPD2 for loading program images.

secondary bootstrap - code that resides in the boot block (block 0) of a medium. This code is loaded and started by the primary bootstrap and in turn loads an starts the XXDP+ monitor.

software table - data structure where DRS stores information regarding operational characteristics of a diagnostic

switch - a modifier for a command

system medium - the medium on the device from which the XXDP+ System was booted

text - a collection of ASCII formatted data consisting of printing characters, tabs, carriage returns and form feeds.

virtual location - a relative memory reference. A program image that has been loaded into the program buffer by UPD2 uses virtual locations; that is, program location 0 is not physical memory location 0, it is the first physical memory location in the program buffer. The XXDP+ monitor does absolute loads and in this case program location 0 is not virtual, but is actually memory location 0.

XXDP+ medium - physical storage, such as a disk pack, MAGtape, cassette, etc., that has been formatted for XXDP+ use.

Appendix H

Component Names

INDEX

A command (XTECO) 102
ADD (DRS) 27
ADR flag (DRS) 34
ASG 57
Assign command . . . . 57 to 58
Autodelete . . . . 44 to 45, 126

Bad address 18
Bad block device 56
BAK 10
Batch control 5, 15, 41
BIC 10, 14
BIN 10, 14
Bit map 55
BOE flag (DRS) 33
BOOT 60, 72
Boot block 56, 60, 73, 126
BOOT command (PATCH) 84
Bootstrap 43, 121, 126
Bootstrap command 60
Bootstrap command (UPD1) 72

C command (XTECO) 95
CCC 10, 15
Chain command 15, 112
Chain file 15
Checksum 18
Checksum error 18
CLEAR command (PATCH) 78
Clear command (UPD1) 67
Clear command (UPD2) 49
Clocks 13
CLR 49, 67
CMI (batch control) 115
Command format conventions . . . 7
Comments (batch control) 115
Component name 6
Conditional execution, batch control 113
Console commands (DRS) 19
Console terminal 61, 126
Console terminal driver 8, 11
Console terminal services 5
CONTINUE (DRS) 25
COPY 56
Copy command 56
Creating text files 92

D command (XTECO) 100
DEC/X11 MAP Files (PATCH) 86

Default device 17
DEL 47, 72
Delete command (UPD1) 72
Delete command (UPD2) 47
DEPO 49, 67
Device CSR address 107
Device driver 126
Device error 18
Device handler 126
DIR 44
Directory 16, 44
Directory command 16, 18
Directory command (UPD2) . . . 44
Directory utility 16
DISPLAY (DRS) 27
DO 57
Do command 57
DRIVER 59
DRIVER command 42
Driver command 59
DROP (DRS) 26
DRS command summary 119
DRS command switches . . . . . 19, 23 to 27, 29
DRS commands 22
DRS commands (batch control) . 112
DRS concepts 19
DRS execution commands 26
DRS flags . . . . 19, 23 to 27, 30, 32
DRS flags summary 120
DRS start-up 19
DRS start-up failure 19
DRS switch combinations 31
DRS switch summary 120
DRS table building 35, 124
DUMP 53, 70
Dump . . . . 49, 52, 67, 69 to 70, 126
DUMP command (UPD1) . . . . . . 69 to 70
DUMP command (UPD2) 51, 53

Edit 92, 126
EDIT command 92
Edit mode (XTECO) 93
Edit mode commands (XTECO) . . 94
Editor 92, 126
Editting text files 92
Enable command 17
End-of-tape command 59
End-of-tape mark 55
EOP switch (DRS) 30
EOT 58
Error halt 13
Error looping (DRS) 33
Error messages (PATCH) 87
Error messages (UPD1) 74

Error messages (UPD2) 63
EVL flag (DRS) 34
EX command (XTECO) 103
EXIT 60
EXIT command (PATCH) 84
Exit command (UPD2) 60
EXIT command (XTECO) 106

FILE 45
FILE command 45
File conventions . . . . 8 to 9
File delete 44
File extension . . . . 9, 14 to 15
File name 9
File rename 44
File transfer 44
Fill character 17
Fill command 16
Fill count 17
FLAGS (DRS) 28
FLAGS switch (DRS) 30
Free blocks 44

GETM command (PATCH) 78
GETP command (PATCH) 82
GOTO (batch control) 114

Halt-on-error (DRS) 32
Hardware table 20, 23, 25, 27, 90, 126
Help command 17
HELP.TXT 17
HICORE . . . . 52 to 53, 69 to 70
High core command (UPD1) 69
High core command (UPD2) 52
HOE flag (DRS) 32
HUDI?? 16

I command (XTECO) 101
IBE flag (DRS) 33
IDR flag (DRS) 34
IER flag (DRS) 33
Indirect command file 43, 57, 121
Invalid filename 18
ISR flag (DRS) 34
IXE flag (DRS) 33

J command (XTECO) 96

K command (XTECO) 100
KILL command (PATCH) 82
KW-11L 13
KW-11P 13

L command (XTECO) 94

LIB 10
Line printer 16, 44
LIST command (SETUP) 91
LOAD 50, 68
Load 49, 126
Load command 15
Load command (UPD1) 68
Load command (UPD2) 50, 56
Load device 13
LOCORE . . . . 52 to 53, 70
LOE flag (DRS) 33
Logical end-of-tape 59
Logical unit number . . . . 57 to 58
Loop-on-error (DRS) 33
LOT flag (DRS) 34
Low core command (UPD1) 70
Low core command (UPD2) 52

Manual intervention (batch control) 115
Medium 126
Memory management 19
Memory size 12
MOD 51, 68
MOD command (PATCH) 78
Modify command (UPD1) 68
Modify command (UPD2) 51
Monitor 5, 11
Monitor command summary 119
Monitor commands (batch control) 112

N command (XTECO) 98
Non-standard CSR address 107
Norewind . . . . 44 to 46, 48, 58, 61

OBJ 10
Operating system 5

Pass 126
Pass (DRS) . . . . 20, 23 to 25, 30
PASS SWITCH (DRS) 30
Patch 5, 49, 127
PATCH command (PATCH) 82
PATCH command summary 121
Physical block 13, 60, 73, 127
Physical location 127
PIP 45
PIP command 45
PNT flag (DRS) 33
PRI flag (DRS) 33
Primary bootstrap 127
PRINT 61
PRINT (batch control) 114
Print command (UPD2) 61
PRINT command (XTECO) 106

PROCEED (DRS) 25
Program buffer 49, 56, 67, 127

QBUS 13
QUIET (batch control) 114
QUIT (batch control) 115

Random access device 55, 109
READ 58
Read command 58
Read-only device driver 11
Read/write device driver . . . 5, 16, 42 to 43, 59, 66, 121
REN 48
Rename command 48
RESTART (DRS) 24
Restart address 12, 42
Run command 14, 112
Runtime services 5, 41

S command (XTECO) 97
Sample file modification . . . 53, 71
Sample XTECO edit 103
SAVE 55, 109
Save commands 55
SAVM 55, 109
SAVP command (PATCH) 82
Secondary bootstrap 11, 13, 56, 127
Sequential access device . . . 55, 109
SETUP 24
SETUP command (SETUP) 90
SMI (batch control) 115
Software table . . . . 20, 24 to 25, 90, 127
START (DRS) 23
Start command 15
Supervisor compatible 5
Switch 127
SYS 10
System device 5, 17, 42
System medium 127

T command (XTECO) 98
TECO 92
TECO command 92
Terminal conventions 8
Test (DRS) . . . . 20, 23 to 25, 29
TESTS switch (DRS) 29
Text 127
TEXT command 92
Transfer address . . . . 14 to 15, 49, 51, 69
Transfer address command (UPD1) 69
Transfer address command (UPD2) 51
Trap catcher . . . . 23 to 25
TXT 10

TYPE 61
TYPE command (PATCH) 81
Type command (UPD2) 61
TYPE command (XTECO) 106

UAM flag (DRS) 34
Unexpected interrupts 23
UNIBUS 13
UNITS switch (DRS) 31
Units-under-test (DRS) 20
UPD1 command summary 120
UPD2 command summary 121
Utility commands (batch control) 112
Utility program 5
UUT . . . . 26 to 27, 31
UUT (DRS) 20, 25

Valid text characters 92
Virtual location . . . . 49, 51, 67 to 68, 127

WAIT (batch control) 115
Wildcard character . . . . 9, 14, 44, 46 to 48, 50, 53, 58, 68, 71

XFR . . . . 51, 53, 69 to 70
XOFF 8
XON 8
XTECO edit command summary . . 122
XTECO non-edit command summary 122
XXDP+ medium 126

ZERO 55
Zero command 55
ZFLAGS (DRS) 28
ZJ command (XTECO) 97

[END OF MANUAL]