TEK SPS BASIC
V02/V02XM
Peripheral Drivers
CP57000/CP57500

Tektronix®

COMMITTED TO EXCELLENCE

Tektronix®
COMMITTED TO EXCELLENCE

TEK SPS BASIC
VØ2/VØ2XM
Peripheral Drivers
CP57000/CP57500

INSTRUCTION MANUAL

Tektronix, Inc.
P.O. Box 500
Beaverton, Oregon 97077

Serial Number _____

First Printing FEB 1980 PRODUCED BY SPS DOCUMENTATION GROUP 070-2740-00

SOFTWARE SUPPORT POLICY

Unless otherwise provided, Tektronix, Inc., agrees that during the one (1) year period following installation, if the customer encounters a problem with this software which the customer's diagnosis indicates is caused by a software defect, the customer may submit a Software Performance Report to Tektronix, Inc. For problems occurring in current, unaltered releases of software, Tektronix, Inc., will respond to Software Performance Reports via a software maintenance periodical. The software maintenance periodical will be provided at no cost to the customer for one year following installation and will contain information for correcting or bypassing verified problems where possible, and will give notice of availability of corrected software.

Any software updates released by Tektronix, Inc., to correct problems during the one (1) year period will be provided to the customer at no charge on the standard distribution media specified in the software documentation. If media other than the standard distribution media is requested, the customer will only be charged for the current cost of the optional media.

SOFTWARE LICENSE

This software product, including subsequent improvements or updates, is furnished under a license for use on a single controller. It may only be copied, in whole or in part (with the proper inclusion of the Tektronix, Inc., copyright notice on the software), for use on that specific controller.

Specification and price change privileges are reserved.

Although the material in this manual has been thoroughly edited and checked for accuracy, Tektronix, Inc., makes no guarantees against typographical or human errors. Also, Tektronix, Inc., assumes no responsibility or liability, consequential or otherwise, of any kind arising from misinterpretation or misuse of the material in this manual. The contents of this manual are subject to change without notice.

PREFACE

This manual describes the peripheral device drivers supported by all releases of TEK SPS BASIC VØ2 and VØ2XM. Any exceptions to an option or a capability of a driver being supported by a specific release of the software are noted where appropriate. Information that pertains only to extended memory (XM) systems is shaded.

The manual is organized by peripheral device type. Each section first shows how to perform the more common operations of a particular type of device using the TEK SPS BASIC peripheral commands. Then the TEK SPS BASIC drivers for devices of that type are described at the end of the section. The first two sections discuss drivers for file-structured devices. Section 1 covers directory-structured devices; Section 2 covers serial-access devices. The third section discusses non-file-structured device drivers. Each of the peripheral commands mentioned in this manual is fully documented in the TEK SPS BASIC VØ2/VØ2XM System Software manual.

TABLE OF CONTENTS

@

SECTION 1

DIRECTORY-STRUCTURED DEVICE DRIVERS

General Operations

Introduction

A directory-structured device is a file-structured peripheral that has a directory of the files stored on it. This directory is a table containing the names of all the files stored on the device and the pointers to where the files are stored. The driver for a specific device accesses a file by searching the device directory for the file name and using the associated pointer to find the actual location of the file. Examples of directory-structured peripherals are hard disk and floppy disk storage devices.

A file is written to a directory-structured device in an integer number of blocks of data. (One block is 256 16-bit words.) The block number is a logical (software) number. The device driver determines the physical location on the device corresponding to that number. Usually, block Ø is the first block of data stored on the device, block 1 is the second, and so on. (An exception to this is the VM driver which makes block 6 the first block.) The device directory always begins at block 6 of the directory-structured peripheral. If the system software can be booted from the device, the absolute loader always begins at block Ø.

In the discussions of the peripheral operations that follow, **the symbol DSn is used to represent the device name and drive number of any directory-structured device** supported by a TEK SPS BASIC VØ2 peripheral driver. When entering any of the examples, be sure to substitute the name (and the drive number when appropriate) of a specific directory-structured device for the DSn symbol (e.g., DX1, DK4, DL2, VM, etc.) The descriptions of each of the TEK SPS BASIC VØ2 directory-structured device drivers that appear at the end of this section include the device name and the number of drives the driver supports.

As each peripheral operation is presented, the appropriate TEK SPS BASIC VØ2 commands are briefly mentioned. For a complete discussion of each command, see Section 4 of the TEK SPS BASIC VØ2 System Software manual.

Loading and Releasing the Device Driver

Before any operations can be performed on a particular peripheral device, the driver for that device must be resident in controller memory. If the directory-structured device being referenced is the same as the system device, the driver for that device is already permanently resident in memory. Otherwise, the driver for that device must be LOADed before the device can be referenced. Assuming that a device driver named "DS.SPS" is stored on the system device, the statement:

```
LOAD "DS.SPS"
```

or simply,

```
LOAD "DS"
```

brings the specified driver into memory. (Notice that a .SPS extension is assumed by the LOAD command.) If at some later time you wanted to delete this driver to free memory space, you could enter:

```
RELEASE "DS"
```

All drivers, except the system device driver and the keyboard driver, are loaded and released in this manner. In the discussions that follow, it is assumed that any required drivers are in memory when the examples execute.

Creating the Directory

As the name implies, a directory-structured device is a file-structured device that has a directory -- a list of the files stored on the medium. As each file is stored or canceled on the medium, the directory is updated to indicate the new or canceled file. The directory also has the location of each file. This allows faster file access than a serial access device. Instead of sequentially searching the medium for the file, the driver searches only the directory for the name and location of the file. Then the file is accessed directly.

Before the medium of a directory-structured device can be used for the very first time, it must be initialized with a ZERO statement to set up space for the directory. This needs to be done only once for a disk, but semiconductor-based extended memory used as a virtual memory peripheral must be initialized each time the system software is loaded. (A disk must

be formatted before it is ZEROed. The FORMAT command formats a DEC RK05 or equivalent disk. However, most disks used by TEK SPS BASIC are preformatted by the manufacturer.)

The directory begins at block 6 of the device. The directory area is variable in length and may range from 1 to 31 directory segments. A directory segment consists of two blocks where a block equals 256 words; thus a segment is 512 words long.

The ZERO command allocates from 1 to 31 directory segments. If no number is specified, the default number of segments, which varies from driver to driver, is used. (See the separate discussion on each driver for its default number of segments.) To set up space for the default number of directory segments use:

ZERO DSn:

To provide for fewer or more directory segments, specify the number after the device name. For example:

ZERO DSn:1Ø

sets aside 1Ø segments for the directory.

The amount of directory space needed depends on the size of the storage area and the number and size of the files to be stored. Theoretically, the maximum number of file entries that can be stored in a device directory is about m times (5Ø7/7) where m is the number of segments allocated by the ZERO command. Due to the way in which the directory is actually filled, however, only (m+1) times (5Ø7/14) entries can be stored, unless you perform a SQUISH command when the directory is first filled. By successfully filling and SQUISHing the device, the theoretical limit can be approached. If a large number of short files are to be stored, at least 2Ø directory segments should be allocated. However, if a smaller number of longer files are to be stored, perhaps the default number of segments is adequate.

If the directory area is overflowed, a fatal error will occur and no more files can be stored on the device. At this point, you should either SQUISH the device or copy any new or existing files onto another device.

Listing the Device Directory

The DIRectory command lists the files stored on a file-structured device. For a directory-structured device, DIR first outputs the device and drive number. Then it lists the file name and the extension, the number of blocks used, and the creation date for each file on the device. (One block equals 256 words.) Free areas are shown in the list as "<unused>." At the end of the listing the total number of free blocks is output. For example:

DIR DSn:

displays the directory on the terminal.

Subsets of the directory can be listed by using the wild card specification (*) in the file name in the DIR statement. For example:

DIR DSn:"*.BAS"

lists all the files on the device that have a .BAS extension, while:

DIR DSn:"PATCH.*"

lists all the patch files stored on the device.

The starting block number (in octal) of the files can be printed also by including the optional keywords WITH BLOCK. For example:

DIR DSn: WITH BLOCK

Instead of a file name, an "<unused>" entry may appear in the directory. The number associated with it is the number of unused blocks in that position on the device. The presence of unused blocks in the middle of a directory indicates that the file previously occupying that position has been CANCELed. Canceling files in the middle of the device is referred to as "fragmenting the device" and usually results in wasted space. However, this wasted space may be reclaimed by executing the SQUISH command, discussed later. It may also be reclaimed by using the INTO option when COPYing or OPENing a file FOR WRITE, also discussed later.

Copying Files

Copies of files can be transferred to or from a file-structured device with the COPY command. For example:

```
COPY DSn:"FILE.BAS" TO DSm:"FILE.BAS"
```

transfers a copy of the file "FILE.BAS" on one directory-structured device to another. The arguments specified on the left of the TO are the source device and file name while the arguments on the right of the TO are the target device and file name. In this example, the source and destination file names were kept the same. However, it is possible to change the file name in the process merely by specifying a new file name in the target argument. Also, if the target is not a file-structured device, no destination file name is required. For example:

```
COPY DSn:"ASCII" TO KB:
```

displays the contents of the ASCII file "ASCII" on the keyboard.

The wild card specification can be used in place of the file name or extension to COPY more than one file. For example:

```
COPY DSn:"*.DAT" TO MT:"*.DAT"
```

transfers a copy of all the files with a .DAT extension on the directory-structured device to magtape.

When COPYing to a directory-structured device, the INTO option may be used to stipulate the maximum number of blocks required by the copied file. For example:

```
COPY DSn:"TEST.DAT" TO DSm:"BACKUP.DAT" INTO 45
```

allots up to 45 blocks on the target device (DSm) for a copy of the file. Notice that the original and the copy will have different file names.

Use of the INTO option with the wild card specification (*) is unnecessary and if used, is ignored. With the wild card notation, as each file is copied, the first sufficient empty space on the target device is used for that file.

When the INTO option or the wild card specification is not used, one half of the largest empty space on the target device is opened for the file. In any case, if the specified or default space exceeds the actual number of blocks used by the file, the unused blocks are returned to empty (unused) status. However, if the specified or default space is less than the number of blocks needed by the file, a fatal error is issued. To avoid this error, use the INTO option or the wild card specification to COPY files whenever one half of the largest empty space may not be large enough for the copied file (such as when a disk is nearly full).

There are circumstances where you may want to copy all the files on a directory-structured device to a blank medium in a directory-structured device. To do this, execute the following SQUISH command:

SQUISH DSn: TO DSm:

This copies all the files from the source device (DSn) to the target device (DSm); the source device remains unchanged, however any files on the target device before the SQUISH are lost.

Canceling Files

Selected files are removed from the device with the CANCEL command. For example:

CANCEL DSn:"SHOOT.BAS","BASIC.DAT"

cancels two files on the same device. The first canceled file is a BASIC program named "SHOOT.BAS"; the second canceled file is a data file named "BASIC.DAT". No default file name extension is provided with the CANCEL command.

When canceling files on the system device, be careful not to cancel any binary TEK SPS BASIC files. These constitute your system software and should not be disturbed. TEK SPS BASIC binary files have a .SPS file name extension, which differentiate them from other files which may be stored on the same device.

Besides creating directory space, the ZERO command can also be used to effectively remove **all** files by reinitializing the device. The ZERO command is discussed under "Creating the Directory."

CAUTION

Be careful when specifying the device argument for the ZERO command. Once a device has been ZEROed, any data stored on it is effectively lost.

Consolidating Free Storage Space

When files are CANCELed, unused areas, which fragment the available free storage space, are left on the device. The SQUISH command consolidates the unused blocks on a directory-structured device into one contiguous area. SQUISH shifts the files on a fragmented device so all the free (unused) blocks follow the last file on the device. To do this simply use:

SQUISH DSn:

Remember, the only time you need to SQUISH a device is when there are canceled files occurring before the last file. If the last recorded file is the only one that has been canceled, you do not need to SQUISH the device. The unused blocks are already at the end of the last file.

CAUTION

The SQUISH command should be used with caution. If it is interrupted, all data on the device may be lost.

Renaming Files

The name of a file stored on a directory-structured device can be changed with the RENAME command. For example:

RENAME DSn:"OLD" TO "NEW"

changes the name of a file from "OLD" to "NEW". RENAME has no default file name extension. If you do not specify an extension in the new file name, the file will not have one.

Storing and Retrieving Data

Sequential-Access Files. Sequential-access files use a method of file access in which the data is stored serially from the beginning of the file and is retrieved in the same order in which it was stored. The general procedure for storing or retrieving data in sequential-access files is to first OPEN a file on the peripheral medium, next to perform the desired operation (READ, WRITE, PRINT, INPUT, READU, or WRITEU), and finally to CLOSE the file. The WRITE and READ commands store and retrieve formatted binary or ASCII data. The PRINT and INPUT commands store and retrieve ASCII strings delimited by carriage returns. The READU and WRITEU commands store and retrieve unformatted binary or ASCII data.

The OPEN command allows access to a new file for storing data or to an existing file for retrieving data. It assigns a peripheral logical unit number (PLUN) to a specified file on the selected device. It also checks that the operation is a legal one and performs any necessary initialization routines. After a file is OPENed, it is addressed by its PLUN not its file name.

A sequential-access file is OPENed either FOR WRITE or FOR READ. Opening a file FOR WRITE creates a new file in which data is stored. Opening a file FOR READ allows the previously stored data to be retrieved.

If you are OPENing a file FOR WRITE, you may use the WITH option. This specifies the number of memory buffers to be used to increase the throughput of the data. (The size of the memory buffer depends on the output device. See the separate discussion on each device driver for the buffer size for that device.) If the WITH option is not specified, just one of these buffers will be used. Increasing the number of buffers often increases the transfer rate, but generally not in direct proportion to the number of buffers specified. In fact, there is a point at which increasing the number of buffers actually decreases throughput due to lack of memory space remaining. The optimum number of buffers largely depends upon the application -- as your own experimentation will verify.

When OPENing a file FOR WRITE on a directory-structured device, you may also use the INTO option. This specifies the number of blocks to open for use by the file. If the INTO option is not used, the driver automatically allocates half of the largest number of contiguous free blocks on the medium. In either case, when the file is CLOSEd, any blocks not written to are returned to an unused status. Normally the default condition does

not cause any problems. However, if you have only a few remaining free blocks on your device, you may want to specify the INTO option. For example, suppose there were 1Ø remaining free blocks on your device and you attempted to write out a file that is 6 blocks long. Since only half of the remaining 1Ø blocks would be allocated, this would cause an error. To get around this problem, use the INTO option with the OPEN command and specify 6 blocks. This will allow you to store the file in its entirety. The INTO option also helps to avoid fragmentation of the medium.

The WITH option and the INTO option, do not apply to an OPEN FOR READ.

After execution of the OPEN command, you can now store or retrieve data. Data is stored by a WRITE, PRINT, or WRITEU statement. Data stored by WRITE is retrieved by the READ command; PRINT, by the INPUT command; and WRITEU, by the READU command. Binary data are stored and retrieved more effeciently with the WRITE and READ commands, while ASCII strings are stored and retrieved more efficiently with the PRINT and INPUT commands. WRITEU and READU write and read files compatible with DEC RT-11 FORTRAN.

After completion of the WRITE, READ, PRINT, INPUT, WRITEU, OR READU operations, the file should be closed. The CLOSE command closes the file to further input or output. It also frees the PLUN for use by another file or device and releases the memory buffers.

Once a sequential-access file that was OPEN FOR WRITE has been CLOSEd, it can no longer be written into. An OPEN FOR READ command is then the only legal OPEN command for that file.

WRITE and READ. The pair of commands used most for data file output and input are WRITE and READ. WRITE was designed to store both numeric and string data in a file; READ was designed to readily retrieve both types of data from the file. With the WRITE/READ pair it is particularly easy to store and retrieve array and waveform data. As an example, consider this partial program which first acquires and stores ten waveforms and then reads and processes them one by one.

With the waveform declared by:

```
1ØØ WAVEFORM WA IS AA(511),DA,HA$,VA$
```

the first step in storing the waveform data is to OPEN a file FOR WRITE. For example:

```
11Ø OPEN #1 AS DSn:"TEST.DAT" FOR WRITE
```

creates a sequential-access file named "TEST.DAT" and assigns it a PLUN of 1. Since the WITH and INTO options are not used, only one memory buffer will be used and the potential size of the file is one half the largest number of contiguous unused blocks on the device. To increase throughput and to insure that the file is large enough for ten waveforms, the following statement could be used instead:

```
11Ø OPEN #1 AS DSn:"TEST.DAT" FOR WRITE WITH 2 INTO 41
```

This alternate line 11Ø provides two memory buffers and OPENs 41 blocks for the file.

The second step in storing a waveform is to write it to the file. A simple statement like:

```
WRITE #1,WA
```

writes all four components of waveform WA to the file known as PLUN 1. Since ten waveforms are to be stored, the WRITE statement can appear in a loop such as:

```
12Ø FOR I=1 TO 1Ø
13Ø GOSUB 1ØØØ\REM ACQUIRE WAVEFORM DATA
14Ø WRITE #1,WA
15Ø NEXT I
```

After all ten waveforms are written to the file, the last step is to close the file. The statement:

```
16Ø CLOSE #1
```

closes the file to further data storage and frees PLUN 1 for use in another OPEN statement. Once closed, the file cannot be reopened for WRITE; it can only be opened FOR READ.

Retrieving the waveform data for processing is just as easy. First, OPEN the file FOR READ. For example:

```
200 OPEN #1 AS DSn:"TEST.DAT" FOR READ
```

opens the file "TEST.DAT" for READ and assigns it the PLUN of 1. Since PLUN 1 is free, the same PLUN is used again; but that is not required. Any free PLUN can be used.

The second step in retrieving the data is accomplished by a statement like:

```
READ #1,WA
```

This reads all four components of the first waveform stored in the file. Repeating this statement ten times reads all ten waveforms. Here the READ statement appears in a FOR loop similar to the loop used for the WRITE operation:

```
210 FOR I=1 TO 10
220 READ #1,WA
230 GOSUB 2000\REM PROCESS WAVEFORM DATA
240 NEXT I
```

The last step is to close the file when done. For example:

```
250 CLOSE #1
```

closes the file and frees the PLUN. Incidentally, since no further data remains to be read, trying to read beyond the end of the data causes an error. However, if you wished to read the data again, instead of closing and reopening the file, you could use a RESET statement. For example, if PLUN 1 is a sequential access file OPEN FOR READ,

```
RESET #1
```

allows the file to be read from its beginning.

WRITEU and READU. WRITEU does not write waveforms and READU does not read them. So, to modify this program for the WRITEU/READU pair, the four components of the waveform must be written individually. Assuming that the units strings HA$ and VA$ are at most ten characters long, line 14Ø could be changed to:

```
14Ø WRITEU #1,AA,DA,HA$=1Ø,VA$=1Ø
```

This writes the array (AA), the data sampling interval (DA), and then the two units strings of the waveform to the file. If either string is shorter than ten characters, spaces are added to the end of the string to make it ten characters long; if either string is longer than ten characters, only its first ten characters are stored.

To read this waveform, line 22Ø should be changed to:

```
22Ø READU #1,AA,DA,HA$=1Ø,VA$=1Ø
```

However, since an HA$ or a VA$ shorter than ten characters is padded with spaces as it is written by the WRITEU in line 14Ø, those added spaces should be removed. Adding the statements:

```
222 HA$=TRM(HA$)
224 VA$=TRM(VA$)
```

trims any trailing spaces from HA$ and VA$ with the TRM string function.

PRINT and INPUT. You would rarely store a waveform in a file with the PRINT command because it would use too much storage space. (To hold just one waveform such as the example, WA, the file would need to be about 15 blocks.) But, if you wanted to PRINT a waveform to a file so that you could COPY it later to a line printer or terminal, use a statement like:

```
PRINT #1,WA
```

However, if the waveform is to be INPUT from the file, each component of the waveform and each element of the array must be terminated by a carriage return (or a comma if the item is a numeric string). PRINT outputs a carriage return at the end of each line. So, the easiest way to delimit each item is to output each item with a separate PRINT statement. Thus,

to modify the example program for the PRINT/INPUT pair, first change the INTO option in line 11Ø to 15Ø blocks. For example:

```
11Ø OPEN #1 AS DSn:"TEST.DAT" FOR WRITE WITH 2 INTO 15Ø
```

This insures that the file is large enough for ten PRINTed waveforms. Then, expand line 14Ø to:

```
14Ø  FOR J=Ø TO 511
141  PRINT #1,AA(J)
142  NEXT J
143  PRINT #1,DA
144  PRINT #1,HA$
145  PRINT #1,VA$
```

which outputs a carriage return after each array element, the data sampling interval, and each units string. This allows the waveform to be read in with a simple INPUT statement such as:

```
22Ø INPUT #1,WA
```

Random-Access Files. Random-access files use a method of file access in which the data can be stored or retrieved, in any order, as logical units called data records. Each data record may consist of one or more data items, but all the data records in a file must be the same length. Where a record is written next does not depend on where a previous record was written in the file. Similarly, which record is read next does not depend on the position of the previously read record. In TEK SPS BASIC VØ2, a random-access file is called a record I/O (input/output) file.

The procedure for storing and retrieving data in a record I/O file is first to create the file with a DEFINE statement, next to OPEN the file FOR UPDATE, then to read or to write the data record with the record I/O form of the WRITEU or READU commands, and finally, to close the file when done.

The first step in using a record I/O file is to create a file of sufficient length on the directory-structured device. The DEFINE command does this and even makes it unnecessary for you to count the number of words or bytes required. You need only to describe the contents of the

record and to specify the number of records desired. The command determines the size of the file by computing the number of bytes per data record and multiplying this by the number of requested records. As the command creates the file on the peripheral, the file is zeroed.

The contents of a data record are described with the keywords ARR, IAR, VAR, and STG. ARR describes a floating-point array (each element is four bytes long; IAR describes an integer array (each element is two bytes long). The expression following ARR or IAR is the number of elements in the array (not the dimension, but the size of the array). VAR describes a single, floating-point variable, while STG describes a string variable. The expression following STG is the number of characters, and therefore the number of bytes, in the string. (The keyword IAR is not supported by DEFINE VØ2-Ø1.)

The total number of bytes in a logical record is calculated from the keyword information. This record length is then multiplied by the number of records requested in the expression following the keyword WITH. This product determines the minimum size of the file. The actual size of the file must be an integer number of blocks (256 words per block). Thus, a file, whose calculated size is 6ØØ bytes (3ØØ words), is really two blocks long.

For example, to define a record I/O file named "TEST.REC" which has 15 records, where each record contains a 512-element array, a variable, and two strings of ten characters each, use:

```
DEFINE DSn:"TEST.REC" AS ARR 512,VAR,STG 1Ø,STG 1Ø,WITH 15
```

After the file is DEFINEd, the next step is to OPEN the file. However, instead of being OPENed for either READ or WRITE, a record I/O file is OPENed FOR UPDATE. This allows both input and output operations and assigns a peripheral logical unit number (PLUN) to the file. For example:

```
OPEN #1 AS DSn: "TEST.REC" FOR UPDATE
```

OPENs the example file as a record I/O file and assigns it PLUN 1.

Once OPENed FOR UPDATE, the file is accessed for output by the record I/O form of the WRITEU command and it is accessed for input by the record

I/O form of the READU command. There is no need to close and reopen the file to change from the output operation to the input operation. As with files OPEN FOR READ or WRITE, record I/O files are referenced by PLUN, not by file name.

The data in a record I/O file is written or read as a record, not as individual data items. Since the records are accessed randomly, the record number must be provided in the WRITEU and READU statements. The record number, which can be an expression, is enclosed in angle brackets (<>) and follows the PLUN. The data records are numbered from zero. So, assuming AA is a 512-element array, to write the first record in the example file "TEST.REC", OPENed as PLUN 1 use:

```
WRITEU #1<Ø>,AA,DA,HA$=1Ø,VA$=1Ø
```

and to write the tenth record use:

```
WRITEU #1<9>,AA,DA,HA$=1Ø,VA$=1Ø
```

Similarly, to read the first record use:

```
READU #1<Ø>,AA,DA,HA$=1Ø,VA$=1Ø
```

and to read the tenth record use:

```
READU #1<9>,AA,DA,HA$=1Ø,VA$=1Ø
```

WRITEU adds trailing blanks to strings that are shorter than the length designated in the WRITEU statement. Thus, any trailing blanks should be removed from strings by using the TRM function after they are input by READU. For example:

```
HA$=TRM(HA$)\VA$=TRM(VA$)
```

When done reading and writing, the last step is to CLOSE the file. For example:

```
CLOSE #1
```

This prevents further access to the file until it is reopened. It also frees the PLUN for use by another file or device.

Random access means that a data record in a record I/O file can be written, read, or updated almost as easily as an array element can be written, read, or updated. In fact, you may think of the record number as being an index into the file and each data record as being an element of the file. Thus, you can use a FOR/NEXT loop to skip through a record I/O file the same way you can use a FOR/NEXT loop to skip through an array. For instance, to read and update every third data record in the example record I/O file, you could use:

```
100 OPEN #1 AS DSn: "TEST.REC" FOR UPDATE
110 FOR I=2 TO 14 STEP 3
120 READU $1<I>,AA,DA,HA$=10,VA$=10
130 GOSUB 1000\REM PROCESS DATA RECORD
140 WRITEU #1<I>,AA,DA,HA$=10,VA$=10
150 NEXT I
160 CLOSE #1
```

One at-a-time, this reads, processes, and rewrites the five data records numbered 2,5,8,11, and 14. Each time the FOR/NEXT loop executes, the record number (I) has a new value, causing a different record to be updated.

Storing and Retrieving Programs

Storing and retrieving programs is a fairly simple process since you do not need to OPEN or CLOSE a file explicitly. As an example, suppose that you wanted to store the following program on a directory-structured device.

```
510  REM PROGRAM TO COMPUTE THE HYPOTENUSE OF RIGHT TRIANGLE
520  PRINT "INPUT LENGTH OF ONE KNOWN SIDE"
530  INPUT X
540  PRINT "INPUT LENGTH OF OTHER SIDE"
550  INPUT Y
560  LET Z=SQR(X*X+Y*Y)
570  PRINT "LENGTH OF HYPOTENUSE IS:",Z
580  END
```

Regular Program Files. If there are no other lines of program text in memory, this program can be stored with this statement:

```
SAVE DSn:"HYPOT"
```

All the program text will be stored in a file called "HYPOT.BAS" on the directory-structured device in drive n. Since no file name extension was specified, an extension of .BAS is assumed by the SAVE command.

Now consider the case where there are other program lines in memory. In this case, the preceding program could be stored with the following statement:

```
SAVE DSn:"HYPOT",51Ø,58Ø
```

which saves only lines 51Ø through 58Ø in the file.

If a program in memory is modified after it has been SAVEd, the new version may be stored instead of the old version with the REPLACE command. It operates similarly to the SAVE command except that it cancels the old file (if there) before saving the new file. For example, if line 58Ø is changed to a RETURN, the statement:

```
REPLACE DSn:"HYPOT"
```

cancels the old "HYPOT.BAS" file and saves all the text in memory in a new file with the same name. (Notice that REPLACE also assumes the .BAS file name extension.) If more lines of text are in memory than just those eight lines,

```
REPLACE DSn:"HYPOT",51Ø,58Ø
```

saves only the lines of text between and including line 51Ø and line 58Ø. Because REPLACE cancels the old file, it can cause the same fragmentation of the media that CANCEL does. (See the discussion, "Consolidating Free Storage Space.")

To retrieve a program, you can use the OLD command. However, you must realize that the OLD command deletes all existing text and variables in memory before loading the program into memory. For example:

```
OLD DSn:"HYPOT"
```

loads all of the program entitled "HYPOT.BAS" from drive n. Alternatively, you could enter:

```
OLD DSn:"HYPOT",51Ø
```

This would have the same effect as the preceeding example, but would also begin program execution at line 51Ø as soon as the program was loaded.

To load a program and delete all the previous text in memory but none of the variables, use the CHAIN command. For example:

```
CHAIN DSn:"HYPOT"
```

deletes all text in memory and then loads the program "HYPOT.BAS", but it does not delete any variables. As with the OLD command, you could also enter:

```
CHAIN DSn:"HYPOT",51Ø
```

which would begin program execution at line 51Ø after loading the program.

Often you may want to load a program into memory without deleting all text or variables. This can be done with the OVERLAY command. For example:

```
OVERLAY DSn:"HYPOT"
```

loads the "HYPOT.BAS" program. No variables are deleted. No text in memory is deleted unless it has the same line numbers as the new program to be loaded, in which case those lines are overwritten.

Notice that OLD, CHAIN, and OVERLAY, which load programs stored by the SAVE or REPLACE command, assume the .BAS file name extension if none is specified.

Fast-Overlay Files. As a program is entered, it is translated into an internal form and stored in the controller memory. SAVE and REPLACE convert the program back to the familiar BASIC language form to store it. Thus, when the program is brought into memory by OLD, CHAIN, or OVERLAY, time is required to translate the text back to the internal form. However, for faster execution of overlaid programs, program segments can be stored with the OVLSAV command and loaded with the OVLOAD command. OVLSAV stores the already translated (internal form) text in a special program file called a fast-overlay file. OVLOAD brings this fast-overlay file back into memory.

For example, to store the previous program in a fast-overlay file use the statement:

```
OVLSAV DSn:"HYPOT"
```

This stores all the program text in memory as a pretranslated fast-overlay file named "HYPOT.BOL" (Notice that OVLSAV has a default file name extension of .BOL.) If more lines of text are in memory than those eight, the lines to be stored can be specified as with SAVE and REPLACE. For example:

```
OVLSAV DSn:"HYPOT",51Ø,58Ø
```

saves only the lines between and including line 51Ø and line 58Ø.

To bring a fast-overlay file back into memory (usually during a running program), use a statement like:

```
1ØØ OVLOAD DSn:"HYPOT"
```

This loads the fast-overlay file "HYPOT.BOL".(OVLOAD also uses the default file name extension .BOL when none is specified.)

Before the fast-overlay file is loaded, any text in memory with line numbers in the range of the line numbers in the fast overlay file is deleted, but no variables are deleted. In this example, any lines in memory with line numbers between and including line 51Ø and line 58Ø are deleted. Also, when the files are loaded, there must be enough free memory available for one input/output buffer, the pretranslated text, and any other information about the text from the file. (The size of the I/O buffer depends on the device used to store the file. See the separate discussion on a particular device driver for the size of the I/O buffer for that device.)

Loading and Releasing SPS Modules

TEK SPS BASIC nonresident commands and drivers are stored in files that have the reserved file name extension, .SPS. These SPS modules are explicitly brought into controller memory from a peripheral device with the LOAD command. (Nonresident commands may also be autoloaded, but only from the system device; drivers are never autoloaded.) When a module is LOADed, it remains resident in memory until it is explicitly deleted from memory by the RELEASE command or until the system software is deleted from memory. (Autoloaded commands are autoreleased if memory space is required for another autoloaded command, data, or an explicitly loaded module.) Both the LOAD and RELEASE commands operate only on files with the .SPS extension and this extension is assumed if the .SPS extension is omitted from the file name specification. For example:

```
LOAD DSn: "VM","OPRINT"
```

loads the two SPS modules, "VM.SPS" and "OPRINT.SPS" while,

```
RELEASE "VM","OPRINT"
```

deletes them from controller memory.

DX Floppy Disk Driver

Attributes

Driver name: DX.SPS

Device name: DX

Device type: Directory-structured

Load module name: SPSDX.LDA

Maximum number of drives: 2 (DXØ and DX1)

Default number of directory segments: 4

Buffer size: 256 words

Maximum number of blocks: 494 (block Ø to block 493)

Medium formatting: Soft-sectored factory formatted

Description

The DX Floppy Disk driver is intended for use with the TEKTRONIX CP115 Dual Drive Flexible Disk unit (Tektronix nomenclature for a Data Systems Design DSO 210 Diskette Memory System), the TEKTRONIX CP112/CP114 Floppy Disk unit (Tektronix nomenclature for the DEC RX11/RXV11 Floppy Disk system), or a similar device. (The CP112 and CP114 are identical except for their interfaces. The CP112 connects to controllers using UNIBUS architecture, while the CP114 connects to controllers using Q-BUS architecture.)

The TEKTRONIX CP115 or CP112/CP114 is a random-access, mass storage device which contains two floppy-disk drives. These drives use preformatted IBM-compatible, flexible diskettes. The diskette can store 256,256 8-bit bytes of information on one side. The outer portion of the recording surface is divided into 77 concentric cylinders (circular "tracks") numbered Ø through 76; Ø is the inner track. Each track is divided into 26 sectors, numbered 1 through 26. Unlike the "hard-sectored" diskettes which contain

a physical hole at the start of each sector, these diskettes use one hole as an index or reference mark for the sectoring. The rest of the surface is "soft-sectored" by a software preformatting routine.

Each sector on the diskette contains two fields: a header field and a data field. The header contains information which indicates the beginning of the header, the track address, the sector address, and a header CRC (Cyclic Redundancy Check). The data field contains a "beginning of data" mark, a 128-byte data space, and a CRC character. For more information on this format, refer to the CP112, CP114, or CP115 manual.

Using the Floppy Disk Unit

Preliminary Instructions. Before using the floppy disk device, check to ensure that it is properly cabled to the system. Also be sure that its power supply is configured for the correct line voltage. (Since the CP112/CP114 do not have power switches, the master control switch for the system controls power to either unit.) In most cases, a Tektronix field engineer will be on site to properly cable and install the system. If you must make changes in the system cabling, consult your field engineer.

Inserting and Removing Diskettes. Each drive is accessed by pressing the pushbutton located on the door of the respective drive. This opens the spring loaded door to the drive. Insert the diskette with the read/write head aperture (Fig. 1-1) facing the drive. Once the diskette has been inserted, close the door to that drive. This will cause the drive spindle to mesh with the registration hole on the diskette and the diskette will now accelerate to a full speed.

To remove a diskette from its drive, open the door to that drive and pull out the diskette. Opening the door disengages the drive spindle so that the diskette may be removed immediately without damage to the spindle or diskette.

CAUTION

Do not open the door to a drive when that drive is being accessed. This can result in data being incorrectly recorded, which causes a CRC error when the sector is being read.

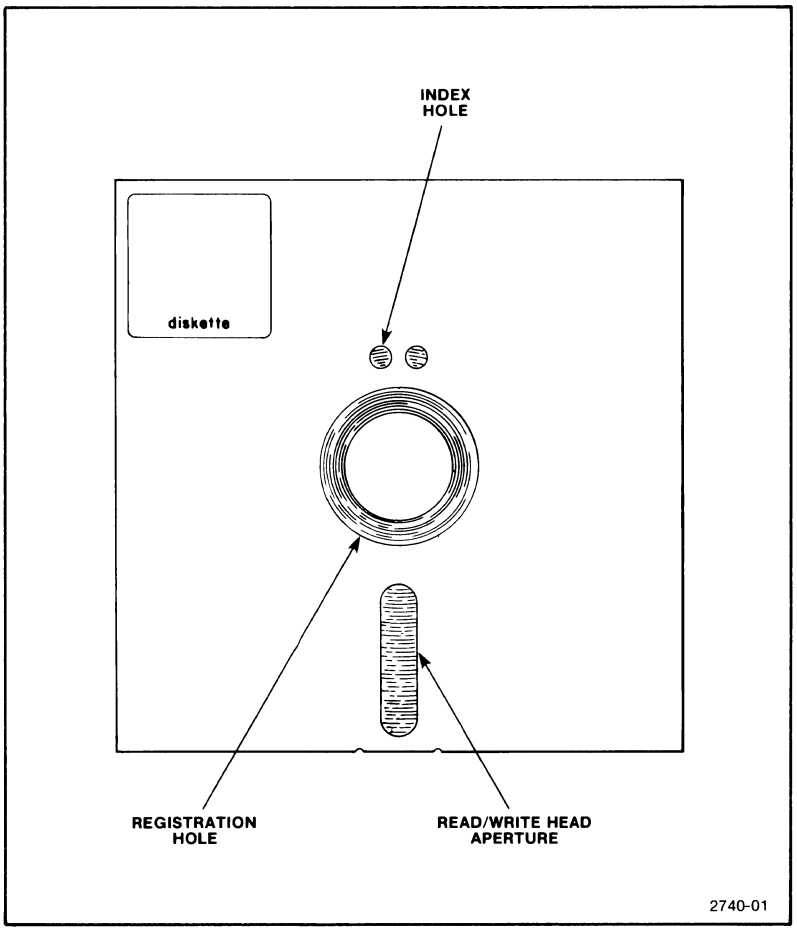

Fig. 1.1. Top view of a floppy disk.

Care and Handling of Diskettes. To prolong the life of diskettes and prevent data read or write errors, the following precautions should be followed:

1. Do not write on the envelope containing the diskette. Write on a label prior to affixing it to the diskette.

2. Do not attach paper clips to the diskette or in any way disfigure it.

3. Do not touch the diskette surface or attempt to clean it.

4. Avoid subjecting the diskette to magnetic fields (such as those found near magnetic tools, terminals, or TV monitors).

5. Avoid exposing the diskette to dust, dirt, excessive heat, or sunlight.

6. Always keep diskettes in their envelopes and store them in a clean, dry place kept at moderate temperature.

How Information is Stored on the Disk

In addition to serving as a medium for loading system software (TEK SPS BASIC operating system software) the device can be used for storing and retrieving BASIC programs and data. The floppy disk is a file-structured device with its own directory. As each new file is stored on the disk, the directory is updated to indicate the name and location of the new file. When a file is canceled, its name is removed from the directory.

The floppy disk has 77 tracks of 26 sectors each. Each sector has a header followed by a 128 byte (64 16-bit word) data field. The numbering of the tracks is from Ø through 76, Ø being the inner track.

When a program or data file is stored on the floppy, it occupies an integer number of blocks -- a block being four sectors on the diskette. Due to the slow speed at which the diskette turns, it is not practical to record on successive adjacent sectors of the diskette. If internal computations caused a delay in writing to the next sector, it would take another entire revolution before the next sector could be accessed. For this reason, every odd sector of track 1 is written into until they are filled. Next, the even sectors of track 1 are written into until they are filled. When all sectors of track 1 have been recorded, track 2 is written into, but the sector number will be displaced six from the last recorded sector on the previous track. Track Ø is not used. This is done to make the diskette IBM and ANSI compatible.

DK Hard Disk Driver

Attributes

Driver name: DK.SPS

Device name: DK

Device type: Directory-structured

Load module name: SPSDK.LDA

Maximum number of drives: 8 (DKØ to DK7)

Default number of directory segments: 8

Buffer size: 256 words

Maximum number of blocks: 48ØØ (block Ø to block 4799)

Medium formatting: Factory formatted or with FORMAT command

Supports DMA (direct memory access)

Description

The DK Hard Disk driver is intended for use with the TEKTRONIX CP110 DISK DRIVE (Tektronix nomenclature for the DEC RK11-DE Disk Driver) or a similar device. The DK driver can support instrument configurations with up to eight disk drives, all controlled by a single disk controller card.

The TEKTRONIX CP110 is a mass storage device consisting of a control module and a moving-head disk drive. It uses removable disk cartridges, similar to the IBM 2315, but with 12 sectors and twice the bit density. Each disk can hold over 2.4 million bytes of data. Data is stored on both sides of the disk by a pair of movable heads. These are always positioned over the opposing tracks simultaneously.

Each side of the disk contains 2Ø3 tracks and each track is divided into 12 sectors capable of storing 256 words (512 8-bit bytes) of information.

Average total access time is 7Ø milliseconds. However, on systems containing more than one disk drive, it may be possible to devise BASIC routines that increase the efficiency.

Each disk sector consists of 13 words of preamble terminating in a sync bit. This is followed by a one-word header, a 256-word data space, and a one-word checksum. The sector is terminated by one word of postamble. A sector pulse signals the beginning of each sector. An index pulse indicates the last sector, signifying that the next sector following is sector Ø. (More information on this subject can be found in the CP110 manual and in the PDP-11 Peripherals Handbook from Digital Equipment Corporation).

Using the Hard Disk Unit

Preliminary Instructions. Before using the hard disk device, check to ensure that it is properly cabled to the system. Also, be sure that its power supply is configured for the correct line voltage. (Since the CP110 does not have its own power switch, the master control switch for the system controls power to the unit.) In most cases, a Tektronix field engineer will be on site to properly cable and install the system. If you must make changes to the system cabling, consult your field engineer.

Controls and Indicators. The CP110 DISK DRIVE includes the following controls and indicators.

RUN/LOAD (rocker switch)	Placing this switch in the Run position locks the drive front door, accelerates the disk to full speed (15ØØ rpm), and loads the read/write heads. Then, the RDY indicator is lit. Placing this switch in the LOAD position unloads the read/write heads, stops the disk rotation, and unlocks the drive front door. Then, the LOAD indicator is lit.

WT PROT (rocker switch)	This momentary-contact switch alternately prevents or allows a write operation on the disk. When the disk is write-protected, the WT PROT indicator is lit and the FAULT indicator is extinguished (if previously lit). When the disk is not write-protected, the WT PROT indicator is not lit.
PWR (indicator)	Lights when the disk drive is powered up.
RDY (indicator)	Lights when the disk is rotating at full speed and the heads are loaded. Also indicates that all interlocks are safe and that a seek, read, or write operation can be performed. It is extinguished when the RUN/LOAD switch is set to the LOAD position.
ON CYL (indicator)	Lights when the drive is in the Ready condition. (A seek or restore operation is not being performed and the read/write heads are positioned and settled.) It is extinguished during a seek or restore operation.
FAULT (indicator)	Indicates a hardware error on the CP110. (See the CP110 manual.) It is extinguished when the WT PROT switch is pressed or when the drive is cycled through a RUN/LOAD sequence.
WT PROT (indicator)	Lights when the disk is write-protected.
LOAD (indicator)	Lights when the read/write heads are fully retracted and the spindle has stopped rotating. It is extinguished when the RUN/LOAD switch is set to the RUN position.
WT (indicator)	Lights during a write operation.
RD (indicator)	Lights during a read operation.

Loading the Disk. The procedure for loading the disk is as follows:

1. Set the RUN/LOAD switch on the disk drive(s) to LOAD and observe that the LOAD indicator lights.

CAUTION

If the LOAD indicator is not lit, the drive front door is locked. In this case, do not attempt to force the door open.

2. Open the front door of the drive and gently insert the disk cartridge fully into the drive mechanism. (The labeled side of the disk should be facing you.) **Do not twist or force the cartridge during insertion.**

3. Close the door of the drive and set the RUN/LOAD switch to RUN. When the RDY and ON CYL indicators are lit, the drive is ready to perform a seek, read, or write operation.

Removing the Disk. The procedure for unloading the disk is as follows:

1. Set the RUN/LOAD switch to LOAD and observe that the RDY indicator goes out. After about 30 seconds, the LOAD indicator will light, signifying that the drive spindle has stopped rotating.

2. Open the front door to the disk drive and gently withdraw the disk cartridge.

3. Close the door to prevent entry of dust or dirt.

Care and Handling of Disk cartridges. To obtain maximum performance and reliability from the disk drive and disk cartridges, the following precautions should be observed.

1. Store cartridges in a clean, dry place away from direct sunlight and excessive heat. Store disks on edge or in stacks of three or four.

2. Unless you are working in a dust free environment, it is recommended that disks be stored in plastic bags. (Since there is only Ø.ØØØ1" clearance between the disk platter and the read/write head, dust, dirt, hair, or fingerprints should never be allowed to contaminate a disk or the disk drive.)

3. Place only stiff cardboard or plastic labels in the molded frame at the front edge of the disk cartridge; do not use any adhesives. Labels placed on any other part of the cartridge may interfere with the drive operation or introduce contamination into the drive or the interior of the cartridge.

4. Allow the temperature of the disk cartridge to become stabilized with the room temperature before using the cartridge.

5. Keep the spindle hub clean and free from nicks and burrs to ensure reliable cartridge operation. Because the hub is slightly magnetic, do not expose it to metal chips that could adhere to the mounting surface. Periodically inspect the hub on the bottom of the cartridge for dirt, metal chips, plastic chips, etc.

6. If during normal operation, you hear a sustained tingling, scratching, or rumbling sound (not to be confused with spindle grounding brushes) shut down the disk drive immediately. Remove the disk cartridge and inspect the read/write heads for damage or excessive dirt. If necessary, clean or replace the heads. Do not reuse the cartridge without first checking for surface damage.

7. Never expose a disk cartridge to strong magnetic fields, such as those surrounding a terminal or TV monitor. This is to prevent information from being erased from the disk.

8. Disk cartridges should be disassembled and cleaned every six months, but more frequent servicing may be required under heavy use. If possible, contact a professional disk cleaning service. In addition to cleaning the cartridge and platter, they will check the bearings and platter for wear and conformance to specified tolerances. If you must clean the disk yourself, carefully disassemble it in a clean room, taking care not to touch the platter with your fingers. Carefully wipe the platter with TEXPADS (Tektronix part no. 006-2398-00, one per order); this is special tissue treated with 99% isopropyl alcohol. After cleaning the platter, reassemble the disk cartridge.

How Information is Stored on the Disk

In addition to serving as a medium for loading system software (TEK SPS BASIC operating system software), the CP110 can be used for storing and retrieving BASIC programs and data. The hard disk is a file-structured device with its own directory. As each new file is stored on the disk, the directory is updated to indicate the name and location of the new file. When a file is canceled, its name is removed from the directory.

The hard disk has 2Ø3 tracks on each side, and each track contains 12 sectors capable of storing 512 bytes (256 16-bit words). Each sector contains 13 words of preamble (terminating in a sync bit), a one-word header, a 256-word data space, and a one-word checksum. The sector is terminated in a one-word postamble. The numbering scheme of the tracks is from Ø to 2Ø2, Ø being the inner track.

The recording scheme for the hard disk is fairly straightforward. Data to be written out to the disk is first stored in a 256-word memory buffer. Thus a block of data on the disk corresponds to exactly one memory buffer or one disk sector. The Absolute Loader is stored beginning at the first sector of track Ø.

Because of the high speed at which the disk revolves (15ØØ rpm), data is recorded sequentially on adjacent sectors of the same track. Normally, data spanning more than one sector can be recorded during a single revolution of the disk. However, occasionally internal computations slow down the data transfer to where adjacent sectors must be recorded on successive revolutions of the disk. In either case, when the upper track (the track on the upper side of the platter) has been completely filled, the track below it will be recorded. When this lower track has been completely filled, the data will be recorded on the next upper track.

DL Hard Disk Driver

Attributes

Driver name: DL.SPS

Device name: DL

Device type: Directory-structured

Load module name: SPSDL.LDA

Maximum number of drives: 4 (DLØ to DL3)

Default number of directory segments: 16

Buffer size: 256 words

Maximum number of blocks: 1Ø24Ø (block Ø to block 1Ø239)

Medium formatting: Factory formatted (cannot be reformatted by software)

Supports DMA (direct memory access)

Description

The DL Hard Disk driver is intended for use with the DEC RL01 disk drive or a similar device. The DL driver can support instrument configurations with up to four RL01 disk drives, all controlled by a single disk controller card.

NOTE

The DL Hard Disk driver is not supported by TEK SPS BASIC VØ2-Ø1.

The RL01 is a mass storage device consisting of a control module and a moving-head disk drive. It uses removable top-loading disk cartridges. Each disk can hold over five million bytes of data. Data is stored on both sides of the disk by a pair of movable heads. These are always positioned over opposing tracks simultaneously.

Each side of the disk contains 256 tracks and each track is divided into 4Ø sectors capable of storing 128 words (256 8-bit bytes) of information. Average total access time is 55 milliseconds. However, on systems containing more than one disk drive, it may be possible to devise BASIC routines that increase the efficiency.

TEK SPS BASIC VØ2 and the Digital Equipment Corp. RT-11 Operating System use the RL01 disk as if there were 2Ø 256-word blocks per track instead of using the 4Ø smaller 128-word data spaces separately. Greater transfer efficiencies and driver compatabilities are achieved in this manner.

Using the Disk Unit

Preliminary Instructions. Before using the RL01 Disk Drive, check to ensure that it is properly cabled to the system. Also, be sure that its power supply is configured for the correct line voltage. In most cases, a Tektronix field engineer will be on site to properly cable and install the system. If you must make changes to the system cabling, consult your field engineer.

Controls and Indicators. The RL01 disk drive includes the following controls and indicators.

LOAD (push-button)

The LOAD button lights to indicate that the cartridge may be loaded or the spindle is stopped.

Pressing the LOAD button locks the drive front door, accelerates the disk to full speed (25ØØ rpm), and loads the read/write heads. Then, the READY indicator is lit.

Pressing the LOAD button again (moving it to the out position) unloads the read/write heads, stops the disk rotation, and unlocks the drive door. The LOAD button light is turned off while the disk is slowing down and lights again when the disk has stopped and the drive door is unlocked.

READY (indicator)	The READY indicator lights when the disk is rotating at full speed and the heads are loaded. When lit, it also indicates that all interlocks are safe and that a seek, read, or write operation can be performed. It is extinguished when the LOAD switch is set to the LOAD position.
FAULT (indicator)	When lit, FAULT indicates a hardware error on the RL01. (Refer to the DEC RL01 Disk Subsystem User's Guide.) It is extinguished when the WRITE PROT switch is pressed or when the drive is cycled through a LOAD sequence.
WRITE PROT (push-button)	This push-button switch alternately prevents or allows a write operation on the disk. When the disk is write-protected, the WRITE PROT indicator is lit and the FAULT indicator is extinguished (if previously lit). When the disk is not write-protected, the WRIT PROT indicator is not lit.

Loading Procedure. The RL01 must be extended from the rack to begin this operation.

1. Raise the cartridge access door (power on).

2. Prepare an RL01 cartridge for loading as follows:

   a. Lift the cartridge by grasping the top cover handle with the right hand.

   b. Support the cartridge with the left hand holding the protection cover.

   c. Lower the top cover handle and push the handle slide to the left with the thumb of the right hand. Again, raise the handle to its full upright position to release the protection cover.

   d. Lift the cartridge from the protection cover and carefully seat it on the drive spindle with the top cover handle recess facing the rear of the machine.

e. Carefully rotate the top cover handle a few degrees clockwise and counterclockwise to ensure that the spindle locating arms are seated properly within the cartridge housing detent slots.

CAUTION

Use care when seating the RL01 cartridge on the drive spindle. Rough handling of the cartridge can cause damage to the spindle/cartridge interface which, in turn, can cause excessive cartridge runout and positioning errors.

f. Gently lower the top cover handle to a horizontal position to engage the cartridge on the drive spindle.

g. Place the protection cover on top of the cartridge.

h. Close the cartridge access door.

Unloading Procedure.

1. Power down the RL01 disk drive as follows:

   a. Press the RUN/STOP switch and wait for the LOAD indicator to light.

   b. Raise the cartridge access door.

2. Remove the RL01 cartridge as follows:

   a. Remove the cartridge protection cover and hold the cover in the left hand.

   b. Push the top cover handle slide to the left before raising the handle.

   c. Raise the top cover handle to a full upright position to release the cartridge from the drive spindle.

   d. Carefully lift the cartridge up and out of the drive and place it in the protection cover.

e. Lower the top cover handle to the horizontal position to lock the protection cover in place.

Care and Handling of Disk Cartridges. To obtain maximum performance and reliability from the disk drive and cartridges, the following precautions should be observed:

1. Store cartridges in a clean, dry place away from direct sunlight and excessive heat.

CAUTION

RL01 disk cartridges must never be stacked on top of each other.

2. Keep cartridges clean.

3. Keep the spindle hub clean and free from nicks and burrs to ensure reliable cartridge operation.

4. Use cartridges at computer room temperature only.

5. Manipulate cartridges by the top cover handle only.

6. When the protection cover is removed (for loading), do not touch disk surfaces, hub center cone, or surfaces.

7. When the protection cover is removed (for loading), interior metal hub surfaces must be clean.

8. When the protection cover is removed (for loading), ensure that the disks are not moved or rotated, since improper disk motion may generate plastic particles which can result in disk damage.

9. When loading or unloading, insert and remove cartridges gently. In addition, **do not use excessive force when manipulating the top cover handle.**

10. If, during operation, a cartridge makes rumbling or continuous tinging sounds, discontinue use of the cartridge immediately. Use of a damaged cartridge on other drives may damage the drives, resulting in additional damage to all other cartridges used thereafter.

11. Each cartridge should be cleaned professionally every six months and/or whenever a specific cartridge is not operating properly.

12. Never expose a disk cartridge to strong magnetic fields, such as those surrounding a terminal or TV monitor. This is to prevent information from being erased from the disk.

How Information is Stored on the Disk

In addition to serving as a medium for loading system software (TEK SPS BASIC operating system software) the RL01 can be used for storing and retrieving BASIC programs and data. The hard disk is a file-structured device with its own directory. As each new file is stored on the disk, the directory is updated to indicate the name and location of the new file. When a file is canceled, its name is removed from the directory.

The disk has 256 tracks on each side, and each track contains 4Ø sectors capable of storing 256 bytes (128 16-bit words). Each sector contains 3 words of preamble, a 3-word header, a 1-word postamble, a 3-word preamble to the data, and 128 words of data. The sector is terminated by a data CRC word and a 1-word postamble. The numbering scheme of the cylinders is Ø to 255, Ø being the outer track.

The recording scheme for the disk is fairly straightforward. Files are stored in an integer number of blocks of data. (One block is 256 16-bit words.) Data to be written to the disk is first stored in a 256-word memory buffer. Thus a block of data on the disk corresponds to exactly one memory buffer or two disk sectors. Block Ø corresponds to sectors Ø and 1 of track Ø, surface Ø. The Absolute Loader is stored beginning at the first sector of track Ø.

Because of the high speed at which the disk revolves (25ØØ rpm), data is recorded sequentially on adjacent sectors of the same track. Normally, data spanning more than one sector can be recorded during a single revolution of the disk. However, occasionally internal computations slow down the data transfer to where adjacent sectors must be recorded on successive revolutions of the disk. In either case, when the upper track (the track on the upper side of the platter) has been completely filled, the track below it will be recorded. When this lower track has been completely filled, the data will be recorded on the next upper track.

VM Virtual-Memory Driver

Attributes

Driver name: VM.SPS

Device name: VM

Device type: Directory-structured

Default number of directory segments: 4

Buffer size: 256 words

Maximum number of blocks: Depends on the amount of extended memory used as a storage peripheral (4 blocks per 1K words)

Description

The Virtual-Memory driver treats extended memory as a peripheral device for storing and retrieving files. This driver is intended for use with any DEC-compatible core or semiconductor memory that is used as an extension of the standard 28K controller (minicomputer) memory. In order for the VM driver to be useful, the system must include DEC KT11 Memory Management hardware. More information on Memory Management can be found in the PDP-11 Processor Handbook published by Digital Equipment Corporation.

In extended memory (XM) systems, all, part, or none of the extended memory can be accessed by the VM driver. The amount of extended memory that can be used as a storage peripheral depends on how the system is initialized at system software load time. By default, all of the extended memory (up to 96K words) is reserved for program data (numeric arrays only). In order to set aside any of the extended memory for use with the VM driver, you must first create your own system parameters file on your system disk by executing the SYSBLD command. After that, each time that the system software is loaded from your disk, your parameters file is used to initialize the system instead of the default parameters.

When you execute SYSBLD, answer the question about the number of 1K word segments to reserve for array storage with a value that is less than the number of 1K word segments available. The difference between the amount available and the amount requested is reserved for use by the VM driver. The minimum size required to allow the LOADing of the VM driver is a single

1K word segment. (See the discussion on the SYSBLD command in Section 4 of the TEK SPS BASIC VØ2 System Software manual for more information.)

In standard memory systems, any extended memory can be used only as a storage peripheral; none can be used for program data.

In general, the VM driver can store or retrieve data at a rate two to ten times faster than the DK driver. However, this is primarily a function of the hardware and thus the improvement in speed varies greatly, depending upon certain conditions. For example, semiconductor memory generally operates considerably faster than core memory, and even the type of semiconductor memory affects speed. Also, the time required to store or retrieve data from the disk varies greatly depending upon the location of the movable disk head in relation to the block of data to be transferred. Thus, it is difficult to give a good "rule of thumb" for the speed advantages of using extended memory as a peripheral device. Other attributes of extended memory are low maintenance, high reliability, and silent operation.

Probably the greatest single disadvantage of using extended memory as a storage peripheral is that data cannot be stored off-line as is the case with a cartridge disk, floppy disk, or magtape reel. In fact, with semiconductor-based extended memory, the data is lost when power is removed from the controller (unless battery backup is used). For these reasons, treating extended memory as a peripheral is primarily useful when you want to quickly store data from an instrument; the data can then be transferred to magtape or a cartridge disk if long-term storage is desired.

How Data is Stored

The extended memory, when used as a peripheral, is a file-structured device with its own directory. As a new file is stored, the directory is updated to record the name and location of the new file. When a file is canceled, its name is removed from the directory.

The recording scheme is straightforward. Data to be written to the extended memory is first stored in a 256-word memory buffer. Then it is written to the extended memory in 256-word blocks. Each 1K of extended memory can hold two blocks of data.

The VM driver assigns block number 6 (the beginning block for the device directory) as the first block in extended memory. This makes block numbers Ø through 5 invalid for the VM driver. Since there is no block Ø, the VM driver cannot permit an absolute loader to be installed, so system software cannot be booted from extended memory.

SECTION 2

SERIAL-ACCESS DEVICE DRIVERS

General Operations

Introduction

A serial-access device is a file-structured peripheral on which the files are stored sequentially with the file name at the beginning of the file rather than in a directory. The driver for a specific device accesses a file by searching the medium linearly (either forward or backward), looking for the file name. Examples of serial-access peripherals are magtape and cassette tape devices.

A file is written to a serial-access device in an integer number of data buffers. (The size of the data buffer depends on the device driver. See the separate discussions on each device driver for the buffer size for that peripheral.) Each new file is written at the logical end of the medium, after the last file on the medium.

In the discussions of the peripheral operations that follow, **the symbol SAn is used to represent the device name and drive number of any serial-access device** supported by a TEK SPS BASIC VØ2 peripheral driver. When entering any of the examples, be sure to substitute the name (and the drive number when appropriate) of a specific serial-access device for the SAn symbol (e.g., MT4, CT1, etc.). The descriptions of each of the TEK SPS BASIC VØ2 serial-access device drivers that appear at the end of this section include the device name and the number of drives the driver supports.

As each peripheral operation is presented, the appropriate TEK SPS BASIC VØ2 commands are briefly mentioned. For a complete discussion of each command, see Section 4 of the TEK SPS BASIC VØ2 System Software manual.

Loading and Releasing the Device Driver

Before any operations can be performed on a particular peripheral device, the driver for that device must be resident in controller memory. Assuming that a device driver named "SA.SPS" is stored on the system device, the statement:

```
LOAD "SA.SPS"
```

or simply

```
LOAD "SA"
```

brings the specified driver into memory. (Notice that a .SPS extension is assumed by the LOAD command.) If at some later time you wanted to delete this driver to free memory space, you could enter:

```
RELEASE "SA"
```

All drivers, except the system device driver and the keyboard driver, are loaded and released in this manner. In the discussions that follow, it is assumed that any required drivers are in memory when the examples execute.

Initializing the Medium

Before using a serial-access medium for the very first time, initialize it with the ZERO command. For example:

```
ZERO SAn:
```

This writes an end-of-tape marker at the beginning of the serial-access device medium in drive n.

Listing the Files Stored on the Device

The DIRectory command lists the files stored on a file-structured device. For a serial-access device, DIR first prints the device and drive

number. Then it lists the file name and the extension plus the creation date for each file on the device. For example:

```
DIR SAn:
```

displays the directory on the terminal.

Subsets of the directory can be listed by using the wild card specification (*) in the file name in the DIR statement. For example:

```
DIR SAn:"*.BAS"
```

lists all the files on the device that have a .BAS extension, while:

```
DIR SAn:"PATCH.*"
```

lists all the patch files stored on the device.

Instead of a file name, an "*EMPTY" entry may appear in the directory. The number associated with it is the number of unused blocks in that position on the device. The presence of empty areas in the middle of a directory indicates that the file previously occupying that position has been CANCELed. On a serial-access device, there is no way to retrieve the space occupied by a CANCELed file unless the medium is ZEROed from that file, forward. See "Canceling Files."

The /F or /R Switch

For a serial-tape device, many commands that specify a source file name may include a forward or reverse switch (/F or /R). The switch specifies the direction of the tape movement when searching for the source file. If the switch is omitted, the tape is rewound before a forward search for the file begins. If the wild card specification (*) is allowed and used in the file name, the switch is ignored.

The peripheral commands that allow the /F or /R switch are:

```
CANCEL
CHAIN
COPY
LOAD
```

OLD
OPEN
OVERLAY
OVLOAD
REPLACE
ZERO

An example of using the /R switch is shown in the "Copying Files" discussion. Examples of using the /F or /R switch with other operations were omitted because the concept is simple and the information would be redundant.

Copying Files

Copies of files can be transferred to or from a file-structured device with the COPY command. For example:

COPY SAn:"FILE.BAS" TO SAm:"FILE.BAS"

transfers a copy of the file "FILE.BAS" on one serial-access device to another. The arguments specified on the left of the TO are the source device and file name while the arguments on the right of the TO are the target device and file name. In this example, the source and destination file names were kept the same. However, it is possible to change the file name in the process merely by specifying a new file name in the target argument. Also, if the target is not a file-structured device, no destination file name is required. For example:

COPY SAn:"ASCII" TO KB:

displays the contents of the ASCII file "ASCII" on the terminal.

The wild card specification can be used in place of the file name or extension to COPY more than one file. For example:

COPY SAn:"*.DAT" TO DX1:" *.DAT"

transfers a copy of all the files with a .DAT extension on the serial-access device to a floppy disk.

When COPYing to a serial-tape device, the forward or reverse switches (/F or /R) may be used to stipulate the direction of the tape movement when searching for the source file. If the switch is omitted, the tape is rewound before a forward search for the file is begun. For example:

```
COPY SAn:/R"TEST.DAT" TO SAm:"BACKUP.DAT"
```

causes the tape to be searched in a reverse direction. (Notice that the original and the copy will have different file names.) If the switch is used with the wild card specification, the switch is ignored.

Canceling Files

Selected files are eliminated from the device with the CANCEL command. For example:

```
CANCEL SAn:"SHOOT.BAS","BASIC.DAT"
```

cancels two files on the same device. The first canceled file is a BASIC program named "SHOOT.BAS"; the second canceled file is a data file named "BASIC.DAT". No default file name extension is provided with the CANCEL command.

When a file is CANCELed on a serial-access device, the file name is changed to "*EMPTY", but no space is reclaimed. Storage space can be regained by ZEROing all or part of the device. For example:

```
ZERO SAn:
```

effectively removes all the files from the serial-access device in drive n by writing an end-of-tape marker at the beginning of the medium. ZERO can also be used to remove all the files from a specified file to the end of the medium. For example:

```
ZERO SAn:"TEST.DAT"
```

logically deletes the file "TEST.DAT" and any files following that file by writing an end-of-tape marker at the beginning of "TEST.DAT".

CAUTION

Be careful when specifying the device argument for the ZERO command. Once a device has been ZEROed, any data stored on it is effectively lost.

Rewinding a Serial-Tape Device

A serial-access device can be returned to the beginning of the tape with the REWIND command. For example:

REWIND SAn:

rewinds the serial-tape device in drive n.

Storing and Retrieving Data

Sequential-Access Files. Sequential-access files use a method of file access in which the data is stored serially from the beginning of the file and is retrieved in the same order in which it was stored. The general procedure for storing or retrieving data in sequential access files is to first OPEN a file on the peripheral medium, next to perform the desired operation (READ, WRITE, PRINT, INPUT, READU, or WRITEU), and finally to CLOSE the file. The WRITE and READ commands store and retrieve formatted binary or ASCII data. The PRINT and INPUT commands store and retrieve ASCII strings delimited by carriage returns. The READU and WRITEU store and retrieve unformatted binary or ASCII data.

The OPEN command allows access to a new file for storing data or to an existing file for retrieving data. It assigns a peripheral logical unit number (PLUN) to a specified file on the selected device. It also checks that the operation is a legal one and performs any initialization routines. After a file is OPENed, it is addressed by its PLUN not its file name.

A sequential-access file is OPENed either FOR WRITE or FOR READ. Opening a file FOR WRITE creates a new file in which data is stored. Opening a file FOR READ allows the previously stored data to be retrieved. Only one file may be OPEN per each drive of a serial-access device at any one time.

If you are OPENing a file FOR WRITE, you may use the WITH option. This specifies the number of memory buffers to be used to increase the throughput of the data. (The size of the memory buffer depends on the output device. See the separate discussion on each device driver for the buffer size for that device.) If the WITH option is not specified, just one of these buffers will be used. Increasing the number of buffers often increases the transfer rate, but generally not in direct proportion to the number of buffers specified. In fact, there is a point at which increasing the number of buffers actually decreases throughput, due to lack of memory space remaining. The optimum number of buffers largely depends upon the application -- as your own experimentation will verify.

The WITH option does not apply to an OPEN FOR READ. (The INTO option is not allowed with serial-access devices.)

After execution of the OPEN command, you can now store or retrieve data. Data is stored by a WRITE, PRINT, or WRITEU statement. Data stored by WRITE is retrieved by the READ command; PRINT, by the INPUT command; and WRITEU, by the READU command. Binary data are stored and retrieved more effeciently with the WRITE and READ commands, while ASCII strings are stored and retrieved more efficiently with the PRINT and INPUT commands. WRITEU and READU write and read files compatible with DEC RT-11 FORTRAN.

After completion of the WRITE, READ, PRINT, INPUT, WRITEU, OR READU operations, the file should be closed. The CLOSE command closes the file to further input or output. It also frees the PLUN for use by another file or device and releases the memory buffers.

Once a sequential-access file has been CLOSEd FOR WRITE, it can no longer be written into. An OPEN FOR READ command is the only legal OPEN command for that file.

WRITE and READ. The pair of commands used most for data file output and input are WRITE and READ. WRITE was designed to store both numeric and string data in a file; READ was designed to readily retrieve both types of data from the file. With the WRITE/READ pair it is particularly easy to store and retrieve array and waveform data. As an example, consider this partial program which first acquires and stores ten waveforms and then reads and processes them one by one.

With the waveform declared by:

```
100 WAVEFORM WA IS AA(511),DA,HA$,VA$
```

the first step in storing the waveform data is to OPEN a file FOR WRITE. For example:

```
110 OPEN #1 AS SAn:"TEST.DAT" FOR WRITE
```

creates a sequential-access file named "TEST.DAT" and assigns it a PLUN of 1. Since the WITH option is not used, only one memory buffer will be used. To increase throughput the following statement could be used instead:

```
110 OPEN #1 AS SAn:"TEST.DAT" FOR WRITE WITH 2
```

This alternate line 110 provides two memory buffers.

The second step in storing a waveform is to write it to the file. A simple statement like:

```
WRITE #1,WA
```

writes all four components of waveform WA to the file known as PLUN 1. Since ten waveforms are to be stored, the WRITE statement can appear in a loop such as:

```
120 FOR I=1 TO 10
130 GOSUB 1000\REM ACQUIRE WAVEFORM DATA
140 WRITE #1,WA
150 NEXT I
```

After all ten waveforms are written to the file, the last step is to close the file. The statement:

```
160 CLOSE #1
```

closes the file to further data storage and frees PLUN 1 for use in another OPEN statement. Once closed, the file cannot be reopened FOR WRITE; it can only be OPENed FOR READ.

Retrieving the waveform data for processing is just as easy. First, OPEN the file FOR READ. For example:

```
200 OPEN #1 AS SAn:"TEST.DAT" FOR READ
```

opens the file "TEST.DAT" FOR READ and assigns it the PLUN of 1. Since PLUN 1 is free, the same PLUN is used again; but that is not required. Any free PLUN can be used.

The second step in retrieving the data is accomplished by a statement like:

```
READ #1,WA
```

This reads all four components of the first waveform stored in the file. Repeating this statement ten times reads all ten waveforms. Here the READ statement appears in a FOR loop similar to the loop used for the WRITE operation:

```
210 FOR I=1 TO 10
220 READ #1,WA
230 GOSUB 2000\REM PROCESS WAVEFORM DATA
240 NEXT I
```

The last step is to close the file when done. For example:

```
250 CLOSE #1
```

closes the file and frees the PLUN. Incidentally, since no further data remains to be read, trying to read beyond the end of the data causes an error. However, if you wished to read the data again, instead of closing and reopening the file, you could use a RESET statement. For example, if PLUN 1 is a sequential-access file OPEN FOR READ,

```
RESET #1
```

allows the file to be read from its beginning.

WRITEU and READU. WRITEU does not write waveforms and READU does not read them. So, to modify this program for the WRITEU/READU pair, the four components of the waveform must be written individually. Assuming that the

units strings HA$ and VA$ are at most ten characters long, line 14Ø could be changed to:

```
14Ø WRITEU #1,AA,DA,HA$=1Ø,VA$=1Ø
```

This writes the array (AA), the data sampling interval (DA), and then the two units strings of the waveform to the file. If either string is shorter then ten characters, spaces are added to the end of the string to make it ten characters long; if either string is longer than ten characters, only its first ten characters are stored.

To read this waveform, line 22Ø should be changed to:

```
22Ø READU #1,AA,DA,HA$=1Ø,VA$=1Ø
```

However, since an HA$ or a VA$ shorter than ten characters is padded with spaces as it is written by the WRITEU in line 14Ø, those added spaces should be removed. Adding the statements:

```
222 HA$=TRM(HA$)
224 VA$=TRM(VA$)
```

trims any trailing spaces from HA$ and VA$ with the TRM string function.

PRINT and INPUT. You would rarely store a waveform in a file with the PRINT command because it would use too much storage space. (To hold just one waveform such as the example, WA, the file would need to be about 15 blocks.) But, if you wanted to PRINT a waveform to a file so that you could COPY it later to a line printer or terminal, use a statement like:

```
PRINT #1,WA
```

However, if the waveform is to be INPUT from the file, each component of the waveform and each element of the array must be terminated by a carriage return (or a comma if the item is a numeric string). PRINT outputs a carriage return at the end of each line. So, the easiest way to delimit each item is to output each item with a separate PRINT statement. Thus, to modify the example program for the PRINT/INPUT pair, expand line 14Ø to:

```
14Ø  FOR J=Ø TO 511
141  PRINT #1,AA(J)
142  NEXT J
```

```
143  PRINT #1,DA
144  PRINT #1,HA$
145  PRINT #1,VA$
```

which outputs a carriage return after each array element, the data sampling interval, and each units string. This allows the waveform to be read in with a simple INPUT statement such as:

```
22Ø INPUT #1,WA
```

Storing and Retrieving Programs

Storing and retrieving programs is a fairly simple process since you do not need to OPEN or CLOSE a file explicitly. As an example, suppose that you wanted to store the following program on a serial-access device:

```
51Ø  REM PROGRAM TO COMPUTE THE HYPOTENUSE OF RIGHT TRIANGLE
52Ø  PRINT "INPUT LENGTH OF ONE KNOWN SIDE"
53Ø  INPUT X
54Ø  PRINT "INPUT LENGTH OF OTHER SIDE"
55Ø  INPUT Y
56Ø  LET Z=SQR(X*X+Y*Y)
57Ø  PRINT "LENGTH OF HYPOTENUSE IS:",Z
58Ø  END
```

Regular Program Files. If there are no other lines of program text in memory, this program can be stored with this statement:

```
SAVE SAn:"HYPOT"
```

All program text will be stored in a file called "HYPOT.BAS" on the serial-access device in drive n. Since no file name extension was specified, an extension of .BAS is assumed by the SAVE command.

Now consider the case where there are other program lines in memory. In this case, the preceding program could be stored with the following statement:

```
SAVE SAn:"HYPOT",51Ø,58Ø
```

which only saves lines 51Ø through 58Ø in the file.

If a program in memory is modified after it has been SAVEd, the new version may be stored instead of the old version with the REPLACE command. It operates similarly to the SAVE command except that it cancels the old file (if there) before saving the new file. For example if line 58Ø is changed to a RETURN, the statement:

```
REPLACE SAn:"HYPOT"
```

cancels the old "HYPOT.BAS" file and saves all the text in memory in a new file with the same name. (Notice that REPLACE also assumes the .BAS file name extension.) If more lines of text are in memory than just those eight lines,

```
REPLACE SAn:"HYPOT",51Ø,58Ø
```

saves only the lines of text between and including line 51Ø and line 58Ø. Because REPLACE cancels the old file, it causes the same empty spaces in the medium CANCEL does.

To retrieve a program, you can use the OLD command. However, you must realize that the OLD command deletes all existing text and variables in memory before loading the program into memory. For example:

```
OLD SAn:"HYPOT"
```

loads all of the program entitled "HYPOT.BAS" from drive n of the serial-access device. Alternatively, you could enter:

```
OLD SAn:"HYPOT",51Ø
```

This would have the same effect as the preceding example, but would also begin program execution at line 51Ø as soon as the program was loaded.

To load a program and delete all the previous text in memory but none of the variables, use the CHAIN command. For example:

```
CHAIN SAn:"HYPOT"
```

deletes all text in memory and then loads the program "HYPOT.BAS" but it does not delete any variables. As with the OLD command, you could also enter:

```
CHAIN SAn:"HYPOT",51Ø
```

which would begin program execution at line 51Ø after loading the program.

Often you may want to load a program into memory without deleting all text or variables. This can be done with the OVERLAY command. For example:

```
OVERLAY SAn:"HYPOT"
```

loads the "HYPOT.BAS" program. No variables are deleted. No text in memory is deleted unless it has the same line number as the new program to be loaded, in which case those lines are overwritten.

Notice that OLD, CHAIN, and OVERLAY, which load programs stored by the SAVE or REPLACE command, assume the .BAS file name extension if none is specified.

Fast-Overlay Files. As a program is entered, it is translated into an internal form and stored in the controller memory. SAVE and REPLACE convert the program back to the familiar BASIC language form to store it. Thus, when the program is brought into memory by OLD, CHAIN, or OVERLAY, time is required to translate the text back to the internal form. However, for faster execution of overlaid programs, program segments can be stored with the OVLSAV command and loaded with the OVLOAD command. OVLSAV stores the already translated (internal form) text in a special program file called a fast-overlay file. OVLOAD brings this fast-overlay file back into memory.

For example, to store the previous program in a fast-overlay file use the statement:

```
OVLSAV SAn:"HYPOT"
```

This stores all the program text in memory as a pretranslated fast-overlay file named "HYPOT.BOL" (Notice that OVLSAV has a default file name extension of .BOL.) If more lines of text are in memory than those eight, the lines to be stored can be specified as with SAVE and REPLACE. For example:

```
OVLSAV SAn:"HYPOT",51Ø,58Ø
```

saves only the lines between and including line 51Ø and line 58Ø.

To bring a fast-overlay file back into memory (usually during a running program), use a statement like:

```
100 OVLOAD SAn:"HYPOT"
```

This loads the fast-overlay file "HYPOT.BOL". (OVLOAD uses the default file name extension .BOL when none is specified.)

Before the fast-overlay file is loaded, any text in memory with line numbers within the range of line numbers in the fast overlay file is deleted, but no variables are deleted. In this example any lines in memory with line numbers between and including line 510 and line 580 are deleted. Also, when the files are loaded, there must be enough free memory available for one input/output buffer, the pretranslated text, and any other information about the text from the file. (The size of the I/O buffer depends on the device used to store the file. See the separate discussion on a particular device driver for the size of the I/O buffer for that device.)

Loading and Releasing SPS Modules

TEK SPS BASIC nonresident commands and drivers are stored in files that have the reserved file name extension .SPS. These SPS modules are explicitly brought into controller memory from a peripheral device with the LOAD command. (Nonresident commands may also be autoloaded, but only from the system device; drivers are never autoloaded.) When a module is LOADED, it remains resident in memory until it is explicitly deleted from memory by the RELEASE command or until the system software is deleted from memory. (Autoloaded commands are autoreleased if memory space is required for another autoloaded command, data, or an explicitly loaded module.) Both the LOAD and RELEASE commands operate only on files with the .SPS extension and assume it if the .SPS extension is omitted from the file name specification. For example:

```
LOAD SAn:"VM","OPRINT"
```

loads the two SPS modules, "VM.SPS" and "OPRINT.SPS" while,

```
RELEASE "VM","OPRINT"
```

deletes them from controller memory.

MT Magtape Driver

Attributes

Driver name: MT.SPS

Device name: MT

Device type: Serial-access

Maximum number of drives: 8 (MTØ to MT7)

Buffer size: 256 words

Description

The MT Magtape driver is intended for use with a TEKTRONIX CP101 magtape system (Tektronix nomenclature for the TM11 magtape system from Digital Equipment Corp.) or a similar device. These systems can control up to eight tape transports (also referred to as drives). Data can be copied from one drive to another; however, it is not possible to have more than one file OPEN per drive at any one time.

TEK SPS BASIC VØ2 does not support magtape as the system device. Thus, the primary use for the magtape medium is storing and retrieving BASIC programs and data.

The magtape systems use industry-compatible, 1/2 inch wide, 9-track tape. (A 1Ø.5 inch diameter reel holds 24ØØ feet of tape; an 8.5 inch diameter reel holds 12ØØ feet of tape.) Data written by a 9-track transport is formatted so as to be industry-compatible. There are eight tracks (or channels) of data. Each track represents a particular bit in the 8-bit byte. Thus, data is stored in bit-parallel, byte-serial fashion. The remaining P track is used to write an odd-parity bit, which will be a "1" in the case of an all-zero data character. The parity functions are performed automatically by the hardware.

Data is recorded on the tape in 512-byte blocks (records) separated by blank areas created as the transport starts and stops. The blank area is referred to as an Inter-Block Gap (IBG) or an Inter-Record Gap (IRG).

Following each block of data, special error detection characters are written. The CRCC (Cyclic Redundancy Check Character) is a coded cyclic parity check character which is developed in a special register while the block is being written. The CRCC character is required for IBM compatibility.

The LRCC (Longitudinal Redundance Check Character) is composed of longitudinal even-parity check bits for each track. The LRCC has odd vertical parity in 9-track transports. Further information can be found in the manual for the magtape transport.

Near the beginning and end of each reel of magtape are reflective markers. These are optically sensed by the transport to indicate the physical beginning and end of the tape, respectively. All tape before the beginning-of-tape (BOT) marker is considered leader. Data recording begins just beyond the BOT mark (load point). The physical end-of-tape (EOT) marker indicates that the end of the tape is approaching.

Magtape File Structure

Magtape files written by the MT driver have the same file structure as files created under the DEC RT-11 Operating System; thus, they use a subset of the VOL1, HDR1, and EOF1 ANSI-standard labels. Each magtape file has the following format:

```
HDR1*---data---*EOF1*
```

where each asterisk represents a tape mark, and HDR1 and EOF1 are the label blocks denoting the beginning and end of file, respectively. Since each ANSI-standard magtape reel begins with a VOL1 label block, a volume (reel) containing a single file has the format:

```
VOL1 HDR1*---data---*EOF1**
```

Similarly, a volume containing two files has the format:

```
VOL1 HDR1*---data---*EOF1*HDR1*---data---*EOF1**
```

A double tape mark following an EOF1 label indicates the logical end of tape. The above formats are illustrated in Fig. 2-1.

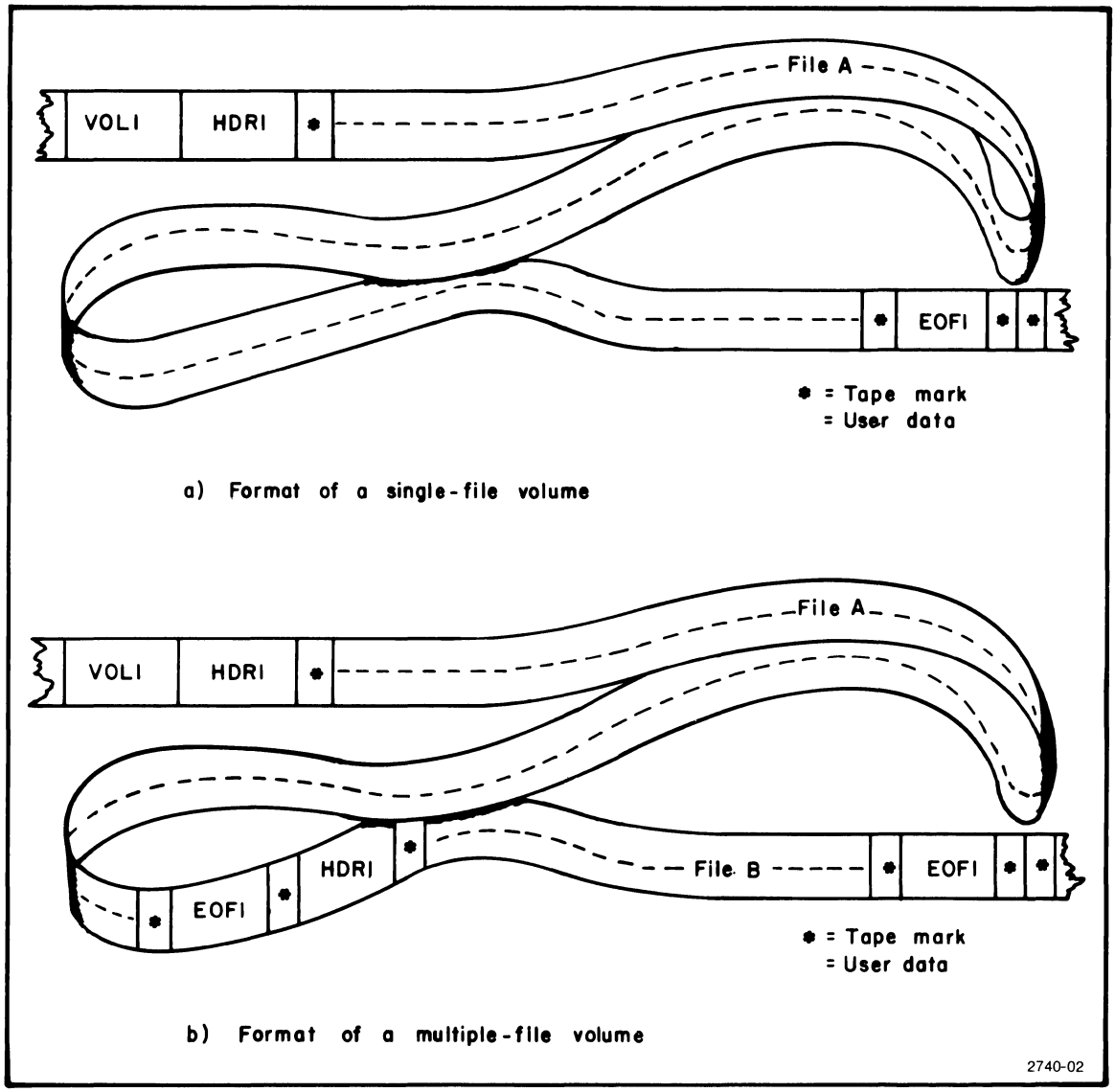

Fig. 2-1. The MT driver supports single-file and multiple-file magtape volumes.

Each label occupies the first 8Ø bytes of a 256-word (512-byte) block, and each byte in the label contains an ASCII character (i.e., if the content of a byte is listed as "1", the byte contains the ASCII code for "1"). Table 2-1 shows the contents of the first 8Ø bytes in the three label blocks. Notice that VOL1, HDR1, and EOF1 each occupy a full 256-word block, of which only the first 8Ø bytes are meaningful. The remaining bytes are blanks.

TABLE 2-1

Contents of the Three MT Label Blocks

Volume-Header Label (VOL1)			
CP[1]	Field Name[2]	Length[3]	Content[4]
1-3	Label identifier	3	VOL
4	Label number	1	1
5-10	Volume identifier	6	RT1101
11	Accessibility	1	Blank
12-37	(Reserved)	26	Blanks
38-51	Owner identifier	14	DD%% used to indicate an RT-11 MT to RSX-11D
52-79	(Reserved)	28	Blanks
80	Label-Standard Version	1	1
81-256	(Unused)	176	Blanks

First File Header Label (HDR1)			
CP	Field Name	Length	Content
1-3	Label identifier	3	HDR
4	Label number	1	1
5-21	File identifier	17	6-character ASCII file name, followed by '.', followed by 3-character ASCII file extension; left justified, remainder of field is blanks
22-27	File Set identifier	6	RT1101
28-31	File Section Number	4	0001
32-35	File Sequence Number	4	0001
36-39	Generation Number	4	0001
40-41	Generation Vsn Number	2	00
42-47	Creation Date	6	␣00000
48-53	Expiration Date	6	blank then 00000
54	Accessibility	1	blank
55-60	Block Count	6	000000
61-73	System Code	13	RT11 left-justified followed by blanks
74-80	(Reserved)	7	blanks
81-256	(unused)	176	Blanks

First End-of-File Label (EOF1)

Same as HDR1 except that the label identifier (CP1-3) is **EOF**, not **HDR**, and the block count field (CP55-60) contains the number of blocks in the file as a decimal value encoded in ASCII characters (for example, if the file was 12 blocks long, the block count field would be 00012).

2740-03

[1]CP - character position in label
[2]Field name - reference name of field
[3]Length - length of field in bytes
[4]Content - content of field in ASCII

TEK SPS BASIC does not support a file structure in which a file spans more than one magtape reel. Thus each reel must contain one or more complete files.

Operating the Magtape Transport

Since the TEK SPS BASIC Magtape driver will work with more than one magtape system, the operating instructions for the magtape transport are not detailed here. However, you are encouraged to read the operating instructions in the applicable magtape manual before attempting to store or retrieve data. It would also be well to read any periodic-maintenance information, such as instructions for cleaning the read-write heads.

CT Cassette Driver

Attributes

Driver name: CT.SPS

Device name: CT

Device type: Serial-access

Maximum number of drives: 2 (CTØ and CT1)

Buffer size: 128 bytes (64 words)

Description

The cassette driver is intended for use with the TEKTRONIX CP100 cassette (Tektronix nomenclature for the DEC TA11 Dual Cassette System) or similar device. TEK SPS BASIC VØ2 does not support cassette tape as the system device. Therefore, the primary use for cassette tape is storing and retrieving data and BASIC programs.

The TEKTRONIX CP100 is a dual-cassette, magnetic tape unit with non-simultaneous drives. Each drive has identical, but separate, motor controls and servo logic. However, both drives use the same selection, formatting and read/write logic. It is not possible to have more than one file OPEN on the same drive at any one time, but it is possible to perform a read or write operation on one drive and to rewind the other drive concurrently. Furthermore, a cassette may be duplicated (copied) by means of the dual-cassette drives.

The CP100 uses Phillips-type cassettes which are specially designed for storage of digital information. Each tape can hold approximately 9Ø,ØØØ bytes (or 45,ØØØ 16-bit words) of information. The data is stored in a single bit-serial track of data. There is no prerecorded timing or format track, so data must be sequentially recorded or retrieved as in a convential magnetic tape system.

The cassette medium is an oxide coated tape with sections of clear leader attached at both ends. The recorded data (stored on the magnetic

portion of the tape) is organized into user-defined collections of data called files. These files are separated from each other by file gaps generated under software control.

Data files in TEK SPS BASIC consist of a 32-byte header record, as many 128-byte data records as needed, and a terminating file gap. Each file is identified by its name, stored in the header record, which consists of one to six characters followed optionally by a decimal point and a one to three character extension (this is the standard SPS file-name format). These names are assigned by the user with the OPEN command. The header record also contains an indication of the file type (e.g., ASCII data, binary data, etc), the number of data records (128 bytes each), and space for additional information not implemented in TEK SPS BASIC. (Additional information on the CP100 and the cassette file structure may be found in the manual for the CP100.)

Operating the Cassette Unit

Preliminary Instructions. Before operating the CP100 CASSETTE tape transport, be sure that its line-cord breaker assembly is configured properly for the line voltage being used. Also, the two BC08-S interconnect cables must be properly connected as mentioned in the CP100 manual. With the unit properly cabled and its line cord connected to a power source, turn on the power switch located on the back of the unit. (Usually this switch will be left on, and the master switch for the system will control power to the unit.) In most cases a Tektronix field engineer will be on site to properly install the system. However, if you must make changes to the system cabling, consult your field engineer or the CP100 manual.

Inserting and Rewinding Cassettes. Before inserting a cassette into the CP100 tape drive, set the cassette write-protect tabs for the desired operation. (See Fig. 2-2.) These tabs are pieces of flexible plastic material that are hinged so that the holes located opposite the tape edge may be covered or exposed as desired. They control the write-protect switches of the tape drive. To inhibit a write operation, fold the tabs back so the cassette recesses are exposed. In some cases, the tabs may be removed to provide permanent protection. To allow a write operation, flip the tabs so the recesses are covered.

The cassette may be inserted with the locking bar in either the opened or the closed position, although it is easier if the locking bar is open.

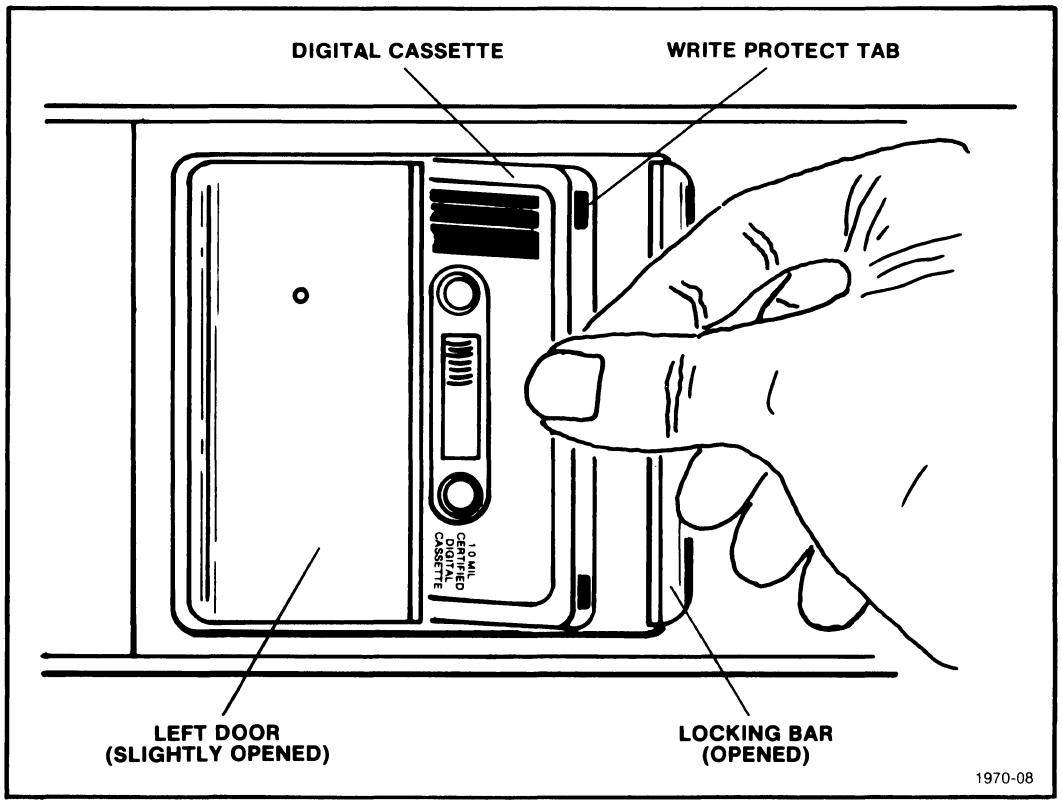

Fig. 2-2. Inserting the cassette.

(See Fig. 2-2.) The locking bar is the bar to the right of the drive mechanism; it is opened by pressing on its flange. The steps for inserting the cassette are as follows:

1. Set the write-protect tabs for the desired operation.

2. Hold the cassette with your thumb and index finger, so that the open side of the cassette faces the read/write heads. Then insert the cassette toward the left (at about a 45 degree angle) into the drive mechanism. The labeled side of the cassette should be facing out and the side which reads "THIS SIDE IN" should face the tape drive.

3. While applying pressure to the left, simultaneously rotate the cassette inward and onto the drive sprockets. This allows the cassette to slide under the door located to the left of the drive mechanism.

4. When the cassette is completely inserted, the locking bar and left door automatically closes flush with the drive mechanism.

5. To remove a cassette, press the locking bar to the right and withdraw the cassette from the drive mechanism.

To rewind a cassette, insert it into either drive and press the REWIND button located to the right of that drive. (See Fig. 2-3.) The cassette should completely rewind to the beginning of the magnetic tape in about 2Ø seconds. If the REWIND button is again pressed, the cassette will rewind to the beginning of the clear leader in about 1 second. When the cassette is completely rewound, the tape will reside on the upper reel. (The cassette can be rewound in the opposite direction by inserting it with the labeled side facing the tape drive.) Once software has been loaded, the cassette may also be rewound by using the REWIND command (e.g., REWIND CT1:).

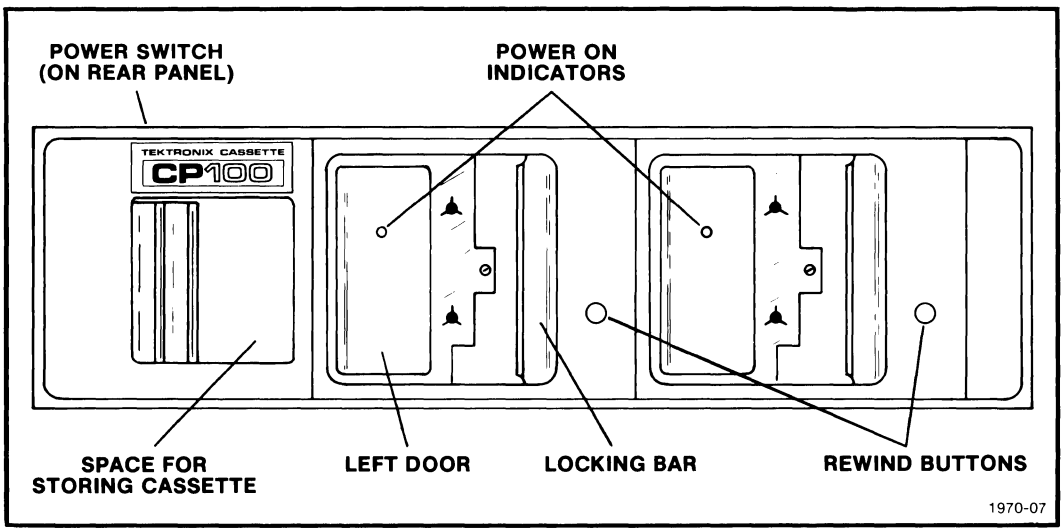

Fig. 2-3. Front panel of a TEKTRONIX CP100 Cassette Tape transport.

Care of Cassettes. It is imperative that only certified digital cassettes be used on the CP100 cassette. These digital cassettes consist of a special heavy base tape which can withstand the high tensions exerted by the CP100. Also, the material used for the head pressure pad is engineered for proper tape stacking. **Never attempt to use an audio cassette in the CP100 tape drive.** The tape in most audio cassettes is not strong enough to withstand the high tensions encountered. Also, their use will result in tape breakage and deterioration, or rapid oxide buildup on the tape heads.

Before using a new cassette (or one that was accidentally dropped), it is best to insert the cassette backwards and completely rewind it. Then turn the cassette over and rewind it again. This ensures that the tape is properly packed onto the cassette and that the full tape reel has the proper operating tension.

It is best to rewind a cassette completely to the beginning of the clear leader to protect the oxide on the tape. Digital cassettes should be protected from dust by storing them in their original plastic boxes. This is especially important since even a speck of dust or dirt on the tape may cause a data error. Also, it is best to store them at room temperature since excessive heat or cold may cause tape deterioration.

SECTION 3

NON-FILE-STRUCTURED DEVICE DRIVERS

Introduction

A non-file-structured peripheral is a device, such as a line printer, paper-tape punch/reader, or the keyboard terminal, on which the data is not referenced by a file name. The three general types of non-file-structured devices supported by TEK SPS BASIC are: data storage and retrieval devices, ASCII output (display) devices, and ASCII input devices. (The keyboard terminal is both an ASCII display and an ASCII input device.) The operations that can be performed by each device type are discussed in the first part of this section. Since each device cannot perform all the operations discussed, a specific device is used in each example. At the end of this section are descriptions of each of the TEK SPS BASIC non-file-structured device drivers.

As each peripheral operation is presented, the appropriate TEK SPS BASIC VØ2 commands are briefly mentioned. For a complete discussion of each command see Section 4 of the TEK SPS BASIC VØ2 System Software manual.

Loading and Releasing the Device Driver

Before any operation can be performed on any peripheral device, the driver for that device must be resident in controller memory. If the non-file-structured device is the keyboard terminal, its driver is made resident when the system software is loaded. Otherwise, the driver must be LOADed before the device can be referenced. Assuming that a device driver named "NFS.SPS" is stored on the system device, the statement:

```
LOAD "NFS.SPS"
```

or simply,

```
LOAD "NFS"
```

brings the specified driver into memory. (Notice that the .SPS extension is assumed by the LOAD command.) When the driver is no longer needed, it can be deleted, freeing memory space, by entering:

```
RELEASE "NFS"
```

Any driver, except the system device driver or the keyboard terminal driver, is loaded and released in this manner. In the discussions that follow, it is assumed that any required drivers are in memory when the examples execute.

NOTE

Do not attempt to LOAD or RELEASE a driver for the keyboard terminal (KB). What will happen if this is done depends upon the release number of the Monitor.

Storage and Retrieval Devices

An example of a non-file-structured data storage and retrieval device is a paper-tape reader/punch. TEK SPS BASIC uses two drivers to support this device. It is treated as two separate devices: one for punching the paper tape and one for reading the paper tape. Each driver may be LOADed or RELEASEd independently as it is needed.

Storing and Retrieving Data

The procedure for storing and retrieving data on a non-file-structured device such as a paper-tape reader/punch is essentially the same as with any SPS peripheral device. First the device must be OPENed FOR WRITE or READ, next the write or read operation must be preformed, and finally the device should be CLOSEd.

For example, suppose that a waveform WA consists of a 512-element array A, a data sampling interval SA, and horizontal and vertical units HA$ and VA$. Such an array could be defined with the command:

```
WAVEFORM WA IS A,SA,HA$,VA$
```

This waveform could be stored on paper-tape with the following routine:

```
1ØØ OPEN #1 AS PP: FOR WRITE
11Ø WRITE #1,WA
12Ø CLOSE #1
```

The command at line 1ØØ OPENs the paper-tape punch (PP) and assigns it peripheral logical unit number (PLUN) 1. Then a 128-null leader is punched

on the tape. (A humanly readable name will also be punched on the tape, if a string expression is specified after the device name, PP.) The WRITE command at line 11Ø outputs the data. Finally, at line 12Ø, a 128-null trailer is generated, and the device is closed.

As with any other WRITE or PRINT operation, multiple buffers can be specified by using the keyword WITH. This can often increase the throughput, depending on the particular application.

To read the stored data back into memory, the tape should be placed in the reader just ahead of the data (but beyond the humanly readable name, if present.) Assuming that waveform WA is defined, the waveform data can be read in with the following routine:

```
200 OPEN #1 AS PR: FOR READ
210 READ #1,WA
220 CLOSE #1
```

Line 2ØØ assigns PLUN 1 to the paper-tape reader as it is OPENed FOR READ. Line 21Ø advances the tape to the end of the leader and begins reading the waveform data. The CLOSE command at line 22Ø closes the device and frees the PLUN for use by another device or a file.

Storing and Retrieving Programs

A BASIC program can be stored on a non-file-structured device such as paper tape with the SAVE command. For example, to store all program text in memory on paper tape, use:

```
SAVE PP:
```

where PP is the device name for the paper-tape punch. This punches a 128-null leader, the program in ASCII format, and a 128-null trailer. (If a program name is specified following the device name, the name is punched on the leader in a humanly-readable form.)

To store only part of the program that is in memory, specify the range of the line numbers to be saved. For example, to save only lines 1Ø through 8Ø, enter:

```
SAVE PP:,10,80
```

A BASIC program is read back into memory from a non-file-structured device such as paper tape with the OLD, CHAIN, or OVERLAY command. OLD

deletes all variables and program text already in memory before loading the program. CHAIN deletes any program text already in memory, but none of the variables, before loading the program. OVERLAY does not delete any variables; it does not delete any text unless some of the program lines already in memory have the same line numbers as some program lines being loaded. Then the old lines are overwritten. For example, to bring in a program stored on paper tape use:

```
OLD PR:
```

or

```
CHAIN PR:
```

or

```
OVERLAY PR:
```

where PR is the device name for the paper-tape reader.

ASCII Output Devices

The keyboard terminal and a line printer are examples of non-file-structured ASCII output devices. The system keyboard terminal is used so often that it is the default output device for several commands. (e.g., PRINT, LIST, and STATUS.) The driver for the keyboard terminal is brought into memory when the system software is loaded; it must never be explicitly LOADed or RELEASEd. The line printer driver must be LOADed before output can be sent to the line printer and should be RELEASEd when no longer in use.

Displaying Data

The easiest way to display the data in controller memory is to PRINT it on the terminal, which is the default device for the PRINT command. For example, assuming that WA is defined as a waveform,

```
PRINT WA
```

outputs the waveform components (the array, the data sampling interval, and the units strings) on the terminal in ASCII characters. (See the discussion on the PRINT command in Section 4 of the System Software manual for the format of the ASCII output.)

NOTE

The Keyboard terminal (KB) is never OPENed or CLOSEd. It is always PLUN Ø. This means that the statement:

```
PRINT #Ø,WA
```

or

```
PRINT #N,WA
```

where N equals Ø, is equivalent to:

```
PRINT WA
```

Before data can be PRINTed on any other device, that device must be explicitly OPENed FOR WRITE. Opening it assigns the peripheral logical

unit number (PLUN) by which the PRINT command references the device. For example, to assign PLUN 1 to the line printer use:

```
OPEN #1 AS LP: FOR WRITE
```

To increase the throughput of the data, more than one buffer can be specified for output by the keyword WITH in the OPEN statement. For example,

```
OPEN #1 AS LP: FOR WRITE WITH 2
```

provides two output buffers for the line printer instead of the default of one buffer.

After the device is OPEN FOR WRITE, the data in controller memory can be output. For example, if WA is a waveform,

```
PRINT #1,WA
```

PRINTs the waveform components on the line printer assigned PLUN 1. The data is printed as ASCII characters in the same format as it is PRINTed on the terminal.

When no more data is to be output on the device, the device should be CLOSEd to free the PLUN. In this case,

```
CLOSE #1
```

closes the line printer and frees PLUN 1 for use by another device or a file.

ASCII data which is stored in a file by the PRINT command may also be displayed on a non-file-structured ASCII output device. For example, the contents of an ASCII file named "ASCII.DAT" stored on a floppy disk in drive 1 is displayed on the line printer by:

```
COPY DX1:"ASCII.DAT" TO LP:
```

(This assumes that both the DX driver and the LP driver are in controller memory when the statement executes.) Similarly,

```
COPY DX1:"ASCII.DAT" TO KB:
```

displays the same file on the terminal.

Listing a Program

Usually, the program in memory is displayed by LISTing it on the terminal, which is the default device of the LIST command. For example:

```
LIST
```

prints all the program text in memory on the terminal while,

```
LIST 1Ø,8Ø
```

prints just lines 1Ø through 8Ø on the terminal.

The program may be LISTed on another non-file-structured ASCII output device by simply including the device name in the LIST statement. For example:

```
LIST LP:
```

LISTs all the program text in memory on the line printer. Similarly,

```
LIST LP:1Ø,8Ø
```

prints just lines 1Ø through 8Ø on the line printer.

ASCII Input Devices

The system keyboard is an example of a non-file-structured ASCII input device. The driver for the keyboard terminal is brought into memory when the system software is loaded; it must never be explicitly LOADed or RELEASEd.

Inputting Data

Program data may be entered directly from the keyboard, which is the default device of the INPUT command. For example:

```
INPUT A,A$
```

allows you to enter the values for A and A$ from the keyboard after the prompting question mark (?) is printed on the terminal by the INPUT command. The number entered for A may be terminated by either a comma or a carriage return; the string entered for A$ must be terminated by a carriage return. (See the INPUT command in Section 4 of the TEK SPS BASIC VØ2 System Software manual for a further discussion of how to input program data from the keyboard.)

NOTE

The keyboard terminal (KB) is never OPENed or CLOSEd. It is always PLUN Ø. This means that the statement:

```
INPUT #Ø,A,A$
```

or

```
INPUT #N,A,A$
```

where N equals Ø, is equivalent to:

```
INPUT A,A$
```

Copying to a File

ASCII data can also be entered from the keyboard and stored directly in a file. For example:

```
COPY KB: TO DX1:"TESTKB.DAT"
```

allows you to enter data from a keyboard and have that data stored in the named file. As with data that is INPUT, numbers entered from the keyword are terminated by either a comma or a carriage return; strings are terminated by a carriage return. Also, since the keyboard is the source device, the COPY command, like the INPUT command, prints a question mark (?) on the terminal to prompt data entry. After each data-terminating carriage return is entered, the COPY command prompts with another question mark. When all the data is entered, the COPY command is terminated by entering a Control-Z. (Data that is stored in a file in this manner is retrieved from the file with the INPUT command.)

LP Line Printer Driver

Attributes

Driver name: LP.SPS

Device name: LP

Device type: Non-file-structured ASCII output device

Maximum number of devices: 1 (LPØ or simply LP)

Buffer size: 132 bytes (characters)

Description

The TEK SPS BASIC LP Line Printer driver supports the TEKTRONIX CP146 or CP445 line printer or a similar device. (The CP146 is Tektronix nomenclature for the LA180 manufactured by Digital Equipment Corporation; the CP445 is Tektronix nomenclature for the TALLY 2215.)

Characteristics of the CP146.

1. Characters per line: 132
2. Character matrix: 7 X 7 dots
3. Printing speed: 18Ø characters per second
4. Character set: 128 (upper and lower case, ASCII)
5. Form width: variable from 3" to 14 7/8"

Characteristics of the CP445.

1. Characters per line: 132
2. Character matrix: 5 X 7 dots
3. Printing speed: 2ØØ lines per minute (upper case)
   165 lines per minute (upper and lower case)

4. Character set: 64 (upper case)
   96 (upper and lower case)

5. Form width: variable from 4" to 14 7/8"

6. Backspace capability: no

7. Character density: 1Ø characters per inch

8. Line density: 6 or 8 lines per inch (switch selectable)
   (10 lines per inch optional)

Preliminary Instructions

Turn on the line printer and ready it for use according to the manual for the line printer being used. You may also want to check for an adequate supply of paper and ribbon in the line printer.

NOTE

If the line printer is not turned on when the OPEN command is executed, the driver will wait until the line printer is ready or a Control-P is entered from the system terminal keyboard. If the line printer is turned off while printing, a fatal error is issued.

How Output is Formatted

A form feed is generated when the line printer is OPENed FOR WRITE or at the start of a LIST operation. As a program is LISTed, it is output in a single, left-justified column, with one program line per line of output. Data is PRINTed on a line printer in the same format that it is output on the terminal, except that some line printers may not respond to certain control characters. See the discussion on the PRINT command in Section 4 of the TEK SPS BASIC VØ2 System Software manual for how each type of data is output by PRINT.

PP and PR Paper-Tape Drivers

Attributes

Driver names: PP.SPS (for punch) and PR.SPS (for reader)

Device names: PP (paper-tape punch) and PR (paper-tape reader)

Device type: Non-file-structured data storage and retrieval device

Maximum number of drives: 1 (PPØ or simply PP)
(PRØ or simply PR)

Buffer size: 64 words (128 bytes)

Description

The paper-tape drivers are intended for use with the TEKTRONIX CP220 Reader/Punch (Tektronix nomenclature for the REMEX RAB6375) or a similar device. TEK SPS BASIC does not support the paper-tape reader/punch as the system device. Thus, the primary use for the paper tape is to store and retrieve BASIC programs and data.

Preliminary Instructions

The CP220 Reader/Punch should be properly cabled to the system and its line cord should be plugged into the master power panel. The CP220 has its own power switch (located to the right of the paper-tape reader mechanism) and this must be turned on before the CP220 can be used. When turned on, a light inside the reader illuminates the front of the unit.

Before using the CP220 for the first time, load a spool of paper tape as explained in the manual for the CP220. Make sure the RUN-LOAD switch is in the RUN position when tape loading is complete. (You can tell when the tape supply is low or when the chad drawer is full; in either case, the PERF STATUS light will illuminate.)

Leader and trailer is automatically generated when punching a program or data tape under TEK SPS BASIC. However, extra leader or trailer may be generated by pressing the FEED/DELETE switch. When in the FEED position,

only sprocket holes are punched. When in the DELETE position, a series of DEL characters is punched. (This later position allows a quick check of the punch mechanism.)

After a tape has been punched or read, it may be rewound by placing the tail of the tape in the spool and by pressing the SPOOL button.

NOTE

If the punch is not turned on when an OPEN FOR WRITE command is executed, the driver will wait until the punch is ready or until a Control-P is entered from the system keyboard terminal. If the punch is turned off while punching, a fatal error is issued.

If the reader is not ready when an OPEN FOR READ command is executed, a fatal error is issued.

When data or a program is to be read, the tape must be positioned in the paper-tape reader such that the sprocket engages the sprocket holes. If a humanly-readable name was punched on the leader, the tape should be positioned past that portion of the leader.

How Output is Formatted

Before the SAVE command stores a program on paper tape or when the paper-tape punch is OPEN FOR WRITE, a 128-null leader is punched on the paper tape. Also, if a file name is provided in the SAVE or the OPEN FOR WRITE statement, the file name is punched on the leader in humanly-readable form. After the SAVE command transfers the program to paper tape or when the paper-tape punch is CLOSEd, a 128-null trailer is punched.

Keyboard Terminal Drivers

Attributes

Driver names: KBG.SPS or KBN.SPS or KBT.SPS or KBE.SPS

Device name: KB

Device type: Non-file-structured data display and input device

Maximum number of devices: 1

Buffer size: 8Ø bytes (79 characters plus a carriage return)

Description

There are four keyboard drivers available in TEK SPS BASIC VØ2.

KBG.SPS. This driver supports graphics capabilities and indicates a Rubout by echoing the deleted character. It shows what character or series of characters is deleted by printing an underscore (_) on either side of the character or series of characters. KBG.SPS is intended for use with a direct-view storage terminal. It is the default keyboard driver.

KBN.SPS. This driver is just like KBG.SPS except that it does not support graphics capabilities. KBN.SPS is intended for use when graphics capabilities are deleted or with typewriter-style terminals that use paper.

KBT.SPS. This driver does not support graphics capabilities. It indicates a Rubout by echoing a backspace. KBT.SPS is intended for use with a raster-scan terminal when graphics capabilities are deleted.

KBE.SPS. This driver supports the high resolution graphics allowed by the Enhanced Graphics Module option of a TEKTRONIX 4014 Computer Display Terminal. It handles a Rubout in the same manner as KBG.SPS. Because KBE.SPS supports the Enhanced Graphics Module option, graphics display commands execute more slowly with KBE.SPS than with KBG.SPS. (KBE.SPS is not supported by TEK SPS BASIC VØ2-Ø1.)

All the drivers can be used with a TEKTRONIX 4010 Series or 4020 Series Computer Display Terminal or similar device. However, the keyboard

terminal drivers for TEK SPS BASIC VØ2-Ø1 do not support the Control-Q or Control-S used to suppress and resume output to terminals that scroll output.

Determining Which Driver

The keyboard terminal driver is brought into memory when the system software is loaded. The keyboard driver that is loaded depends on the existence of a system-parameters file on the disk from which the software is booted. If no such file exists, the default driver, KBG.SPS, is loaded. If the file exists, the keyboard driver specified in the file is loaded. Executing the SYSBLD command creates (or changes) the system-parameters file. This allows another keyboard driver to be loaded the next time the system software is booted.

You can obtain the name of the current keyboard driver by typing:

```
STATUS
```

This command reports the status of the system, including the name of the keyboard driver in memory. To change to another keyboard driver, execute SYSBLD and reboot BASIC. (See the discussion on the SYSBLD command in Section 4 of the TEK SPS BASIC VØ2 System Software manual.)

Special Function Keys

The keyboard drivers support these special function keys. A control character is entered by holding down the control key while striking the desired character key. (When a control character is entered, the driver echos an up arrow (^) followed by the character.)

Key	Action
Return	Inserts a carriage return into the keyboard buffer and makes the contents of the buffer available to the Monitor. A carriage return, line feed is echoed.
Rubout	If the keyboard buffer is not empty, deletes the last character inserted into the buffer.

Control-O	Alternately suppresses and allows the display of most output sent to the terminal. Error messages are not suppressed. The first Control-O suspends display; the second, resumes it. Any output directed to the terminal between the suspension and resumption of the display is lost.
Control-P	Terminates a program and returns BASIC to idle mode.
Control-Q	Resumes terminal output that was suspended by a Control-S. No data is lost. (Control-Q is not supported by TEK SPS BASIC VØ2-Ø1.)
Control-S	Suspends terminal output until a Control-Q is entered. No data is lost. (Control-S is not supported by TEK SPS BASIC VØ2-Ø1.)
Control-U	Deletes any characters from the keyboard buffer. (Deletes the line being entered.)
Control-Z	Terminates input to the COPY command when the terminal is the source device.

APPENDIX A

LOADING TEK SPS BASIC

In order to boot TEK SPS BASIC from a peripheral device, the proper SPS load module of the operating system (the .LDA file) for that device must be on the medium. (Not all peripherals are supported by an SPS load module. Check the individual discussions on each device driver in this manual to see if an SPS load module exists and what its name is.) Also, before the software can be booted, an absolute loader must have been installed on the medium by either the HOOK or HOOKQ command. (The absolute loader is a stand-alone program which, in this case, loads the operating system. The absolute loader is brought into memory by the bootstrap program.) If you follow the archiving procedure in Appendix C to make working copies of the TEK SPS BASIC System software, the correct absolute loader is installed. See the discussions on the HOOK and HOOKQ commands in Section 4 of the TEK SPS BASIC VØ2 Systems Software manual for more information.

The device name and drive number from which BASIC is loaded becomes the **system device**. This is the device and drive from which commands are autoloaded. It is also the default device and drive for many of the peripheral commands (e.g., BOOT, CANCEL, COPY, DIR, OLD, OPEN, OVERLAY, OVLOAD, READ, SAVE, WRITE, etc.) The system device driver is loaded with the operating system; it is included in the SPS .LDA file for that device.

After the hardware system is properly connected and powered-up, insert the medium with your copy of TEK SPS BASIC into the device for that medium. If the device has more than one drive, use the drive you prefer.

Next, follow the bootstrap procedure for the ROM bootstrap card in your controller. Some common hardware bootstrap procedures are briefly discused in Appendix B.

When the ROM bootstrap program of the controller issues its prompt, enter the device name and the drive number from which BASIC will be loaded. For example, if your software is on a DK hard disk in drive 1, enter:

```
DK1
```

Now, depending on which absolute loader has been installed on the medium for your copy of BASIC, one of three things will happen.

1. If the SPS absolute loader was installed by the HOOK command, TEK SPS BASIC is loaded automatically. This is the most common situation.

2. If the DEC RT-11 absolute loader was installed by the HOOK command, the DEC RT-11 Operating System is loaded. To load TEK SPS BASIC, enter:

```
RUN LOADER
```

in response to the RT-11 prompt, a dot (.). When LOADER prints its prompt, an asterisk (*), enter the name of the SPS .LDA file for your device. (Do not type the .LDA extension.) For example, if the device is a DK hard disk, enter:

```
SPSDK
```

NOTE

To return to the DEC RT-11 Operating System from TEK SPS BASIC, enter:

```
BOOT
```

This reboots the device with DEC RT-11 as the operating system.

3. If the LDA absolute loader has been installed by the HOOKQ command, any file with the .LDA extension can be loaded. To load TEK SPS BASIC, enter the name of the SPS .LDA file for your device in response to the prompting asterisk (*). (Do not enter the .LDA extension.) For example, if the device is a DK hard disk, enter:

```
SPSDK
```

After BASIC is loaded, the version and release numbers of the BASIC Monitor and the amount of free controller memory are printed on the terminal. Finally,

```
READY
*
```

is displayed to show that TEK SPS BASIC is loaded and waiting for you to enter program text or an immediate mode command.

APPENDIX B

STANDARD HARDWARE BOOTING PROCEDURES FOR TEK SPS BASIC VØ2

M9301 Bootstrap ROM Card

A. Perform either 1 or 2 below.

1. On a controller without switch registers:

   Press CNTRL-BOOT.

2. On a controller with switch registers:

   a. Press HALT.
   b. Enter the bootstrap address
      (usually either 173ØØØ or 173Ø1Ø) on the switch registers.
   c. Press LOAD ADDRESS.
   d. Press ENABLE.
   e. Press START.

B. In response to the prompt character ($) printed on the terminal, type the device name and drive number, followed by a carriage return. The devices supported by the M9301 and TEK SPS BASIC VØ2 are:

DXn where n is a Ø or 1

DKn where n is an integer between Ø and 7, inclusive.

M9312 Bootstrap ROM Card

A. Perform either 1 or 2 below.

1. On a controller without switch registers:

   Press CNTRL-BOOT.

2. On a controller with switch registers:

   a. Press HALT.
   b. Enter the bootstrap address
      (usually either 173ØØØ or 173Ø1Ø) on the switch registers.
   c. Press LOAD ADDRESS.
   d. Press ENABLE.
   e. Press START.

B. In response to the prompt character ($) printed on the terminal, type the device name and drive number, followed by a carriage return. The devices supported by the M9312 (with the required ROM) and TEK SPS BASIC VØ2 are:

DXn where n is a Ø or 1

DKn where n is an integer between Ø and 7, inclusive.

DLn where n is an integer between Ø and 3,inclusive
(not supported by TEK SPS BASIC VØ2-Ø1)

DYn where n is a Ø or 1
(not supported by TEK SPS BASIC VØ2-Ø1)

Standard ROM Bootstrap on SBT Module in CP4165

A. Press RESTART.

B. In response to the prompt (DEV=) printed on the terminal, type:

DXn where n is a Ø or 1

Do not enter a carriage return.

APPENDIX C

ARCHIVING YOUR SOFTWARE

We strongly urge you to create working copies of your software as soon as possible and to keep your original copy as an archive. We also recommend that you never write on your archive-copy medium and if possible, that you always write-protect your archive software when making working copies.

To assist you in archiving your software, some methods for creating working copies are discussed here. The instructions are grouped under two general headings: hard-disk based systems and floppy-disk based systems. Under the heading for your system, find the example that best describes your new software and follow the instructions on how to archive it. The examples include:

Hard-Disk Based Systems

1. System software (without instrument checkout software) on hard disk.

2. System software with instrument checkout software on hard disk.

3. Separate package or module on hard disk.

4. Separate package or module on floppy disk.

5. Instrument checkout software on floppy disk.

Floppy-Disk Based Systems

1. System software on a single floppy disk.

2. TEK SPS BASIC on minimum number of floppy disks.

3. Separate package or module on floppy disk.

4. Instrument checkout software on floppy disk.

After you have made a copy of your software, check to see if it requires patching. Do this by looking in the issue of the SPS Programming Update shipped with your software. Included in this publication are all the reported software errors and patches. Look through the list of patches and the descriptions of the errors they fix. If any of the patches for your version and release of the software are ones you want to implement, carefully follow the patching directions in the SPS Programming Update. Patch the working copy of the software you just made. **Do not patch the archive software.** When you have finished copying and patching your software, store the issue of the SPS Programming Update with your archive software.

NOTE

If you did not receive an issue of the SPS Programming Update with your software, in the U.S.A. request one by writing:

SPS Programming Update
Group 157 (94-384)
Tektronix, Inc.
P.O. Box 500
Beaverton, OR 97077

Outside the U.S.A., contact your local Tektronix representative.

If you do any patching, you may want to save a copy of the patch files created when PATCH.BLD was run. (Such a file has a numeric file name extension, e.g. "PATCH.ØØ1".) **Do not copy these files onto your archive medium.** Instead, copy these patch files onto a separate disk or tape and store this separate medium with the archive software.

Hard-Disk Based Systems

In the discussions that follow, the example device is the DK type of hard disk (a DEC RK05 or similar device). To make working copies of the software on another type of hard disk supported by TEK SPS BASIC, substitute the proper device name for all occurrences of the DK device name, shown in bold.

System Software (without Instrument Checkout Software) on Hard Disk

If you have purchased TEK SPS BASIC system software on hard disk, you have two ways to make a working copy of your software.

I. SQUISH to a blank disk. This method is simple but transfers more files to your working copy than you need for a hard-disk system.

1. Load the original disk into drive Ø, bootstrap, and **write-protect.**

2. Load a blank, formatted disk into drive 1.

3. When the system is loaded and READY is printed on the terminal, type:

```
SQUISH DK: TO DK1:
HOOK DK1:
```

4. When READY is printed on the terminal, remove the original disk from drive Ø. This should be stored as the archive copy.

5. The disk in drive 1 is now your working copy of TEK SPS BASIC.

II. COPY selected files. With this method, you transfer only the files you need for a hard-disk system onto your working copy.

1. Follow steps 1 and 2 in **I** above.

2. When the system is loaded and READY is printed on the terminal, type:

```
ZERO DK1:
COPY "*.SPS" TO DK1:"*.SPS"
COPY "*.OVL" TO DK1:"*.OVL"
COPY "PATCH.*" TO DK1:"PATCH.*"
COPY "SPSDK.LDA" TO DK1:"SPSDK.LDA"
HOOK DK1:
```

3. Follow steps 4 and 5 in **I** above.

System Software with Instrument Checkout Software on Hard Disk

If you have purchased TEK SPS BASIC system software with instrument checkout software on a single hard disk, use one of these procedures to create working copies of this software. The type of the checkout software determines which method you use.

I. BASIC checkout software. Use this procedure if the checkout software is a BASIC program (e.g., CP56008 7912AD Checkout Software).

1. Load the original disk into drive Ø, bootstrap, and **write-protect.**

2. Load a blank, formatted disk into drive 1.

3. When the system is loaded and READY is printed on the terminal, type:

```
SQUISH DK: TO DK1:
HOOK DK1:
```

4. When READY is printed on the terminal, remove the original disk from drive Ø. This should be stored as the archive copy.

5. The disk in drive 1 is now your working copy of TEK SPS BASIC with the instrument checkout software.

II. Stand-alone checkout software. Use this procedure if the checkout software is a stand-alone software with its own .LDA file (e.g., CP56001 P7001/R7912 Checkout Software).

1. Follow steps 1 and 2 in **I** above.

2. When the system is loaded and READY is printed on the terminal, type:

```
SQUISH DK: TO DK1:
HOOKQ DK1:
```

3. Follow steps 4 and 5 in **I** above.

When the working disk is bootstrapped, a prompt (*) will appear on the terminal. Any file with the .LDA extension can then be loaded by entering the file name without the file name extension.

To load and execute TEK SPS BASIC, type:

```
SPSDK
```

To load and execute the instrument checkout software, type the name of the .LDA file, but without the .LDA extension.

Separate Package or Module on Hard Disk

If you have purchased a separate software package or supplemental module to add to your system, you have two options when making a working copy.

I. SQUISH to a blank disk. Maintain a separate disk as a working copy of the package or module.

1. Boot TEK SPS BASIC from drive Ø. When the system is loaded and READY is printed on the terminal, type:

```
LOAD "SQUISH"
```

2. Remove the disk with TEK SPS BASIC from drive Ø.

3. Load the original disk with the package or module into drive Ø and **write-protect.**

4. Load a blank, formatted disk into drive 1.

5. Type:

SQUISH **DK**: TO **DK**1:

6. When READY is printed on the terminal, remove the original disk from drive Ø. This should be stored as the archive copy.

7. Use the disk in drive 1 as the working copy of the package or module.

II. COPY to working disk. Add the package or module to your working copy of TEK SPS BASIC on hard disk.

1. Load your working copy of TEK SPS BASIC into drive Ø, bootstrap, but do not write-protect.

2. Load the original disk with the package or module into drive 1 and **write-protect.**

3. When the system is loaded and READY is printed on the terminal, type:

COPY **DK**1:"*.SPS" TO "*.SPS"

4. When READY is printed on the terminal, remove the original disk from drive 1. This should be stored as the archive copy.

5. Your working copy in drive Ø now includes the new package or module.

Separate Package or Module on Floppy Disk

If you have purchased a separate software package or supplemental module on floppy disk to add to your hard-disk system, you must have a floppy-disk device. We assume here that you want to add the package or module to your working copy of TEK SPS BASIC on hard disk.

1. Load your working copy of TEK SPS BASIC into hard-disk drive Ø, bootstrap, but do not write-protect.

2. Load the original disk with the package or module into floppy-disk drive Ø and **write-protect** (if possible).

3. When the system is loaded and READY is printed on the terminal, type:

```
LOAD "DX"
COPY DX:"*.SPS" TO DK:"*.SPS"
```

4. When READY is printed on the terminal, remove the original disk from floppy-disk drive Ø. This should be stored as the archive copy.

5. Your working copy in hard-disk drive Ø now includes the new package or module.

Instrument Checkout Software on Floppy Disk

If you received instrument checkout software on a floppy disk, use one of these procedures to add it to your hard-disk system. The type of the checkout software determines which method you use. To transfer the software to your working copy on hard disk, you must have a floppy disk device.

I. BASIC checkout software. Follow this procedure to copy checkout software that is a BASIC program (e.g., CP56008 7912AD Checkout Software).

1. Load your working copy of TEK SPS BASIC into hard-disk drive Ø, bootstrap, but do not write-protect.

2. Load the original disk with the checkout software into floppy-disk drive Ø and **write-protect** (if possible).

3. When the system is loaded and READY is printed on the terminal, type:

```
LOAD "DX"
COPY DX:"*.*" TO DK:"*.*"
```

4. When READY is printed on the terminal, remove the original disk from floppy-disk drive Ø. This should be stored as the archive copy.

5. Your working copy in hard-disk drive Ø now includes the checkout software.

II. Stand-alone checkout software. Follow this procedure to copy checkout software that is stand-alone software with its own .LDA file (e.g., CP56001 P7001/R7912 Checkout Software).

1. Follow steps 1 through 3 in **I** above.

2. Then enter:

```
HOOKQ DK:
```

3. Follow steps 4 and 5 in **I** above.

Now when the working disk is bootstrapped, a prompt (*) will appear on the terminal. Any file with an .LDA extension can then be loaded by entering the file name, without the file name extension.

To load and execute TEK SPS BASIC, type:

```
SPSDK
```

To load and execute the instrument checkout software, type the name of the .LDA file, but without the .LDA extension.

Floppy-Disk Based Systems

In the discussions that follow, the example device is the DX type of floppy disk (a preformatted, IBM-compatible, single-density flexible diskette). To make working copies of the software on another type of floppy disk supported by TEK SPS BASIC, substitute the proper device name, shown in bold.

System Software on a Single Floppy Disk

If you have purchased TEK SPS BASIC system software on floppy disk, you have two ways to make a working copy of your software.

I. **SQUISH to a blank disk.** This method is simpler but transfers more files to your working copy than you need for a floppy-disk system.

1. Load the original disk into drive Ø, bootstrap, and **write-protect** (if possible).

2. Load a blank, formatted floppy disk into drive 1.

3. When the system is loaded and READY is printed on the terminal, type:

```
SQUISH DX: TO DX1:
HOOK DX1:
```

4. When READY is printed on the terminal, remove the original disk from drive Ø. This should be stored as the archive copy.

5. The disk in drive 1 is now your working copy of TEK SPS BASIC.

II. **Copy selected files.** With this method, you transfer only the files you need for a floppy-disk system onto your working copy.

1. Follow steps 1 and 2 in I above.

2. When the system is loaded and READY is printed on the terminal, type:

```
ZERO DX1:6
COPY "*.SPS" TO DX1:"*.SPS"
COPY "*.OVL" TO DX1:"*.OVL"
COPY "PATCH.*" TO DX1:"PATCH.*"
COPY "SPSDX.LDA" TO DX1:"SPSDX.LDA" INTO 95
HOOK DX1:
```

3. Follow steps 4 and 5 in **I** above.

TEK SPS BASIC on Minimum Number of Floppy Disks

1. Load the original disk with the proper monitor file (SPS**DX**.LDA) into drive Ø, bootstrap, and write protect (if possible.)

2. Load a blank, formatted disk into drive 1.

3. When the system is loaded and READY is printed on the terminal, type:

```
LOAD "SQUISH"
SQUISH DX: TO DX1:
HOOK DX1:
```

4. When READY is printed on the terminal, remove the original disk from drive Ø. This should be stored as the archive copy.

5. Remove the disk from drive 1. It is now your working copy of TEK SPS BASIC System software.

6. Load another of the original disks to be archived into drive Ø and **write-protect** (if possible).

7. Load another blank, formatted disk into drive 1.

8. Type:

```
SQUISH DX: TO DX1:
```

9. When READY is printed on the terminal, remove the original disk from drive Ø. This should be stored as the archive copy.

10. Remove the disk from drive 1. It is now your working copy of additional SPS software.

11. Repeat steps 6 through 10 until all the original disks are archived.

Separate Package or Module on Floppy Disk

If you have purchased a separate software package or supplemental module to add to your system, your have two options when making a working copy.

I. SQUISH to a blank disk. Maintain a separate disk as a working copy of the package or module.

1. Boot TEK SPS BASIC from drive 1. When the system is loaded and READY is printed on the terminal, type:

```
LOAD "SQUISH"
```

2. Remove the disk with TEK SPS BASIC from drive Ø.

3. Load the original disk with the package or module into drive Ø and **write-protect** (if possible).

4. Load a blank, formatted disk into drive 1.

5. Type:

```
SQUISH DX: TO DX1:
```

6. When READY is printed on the terminal, remove the original disk from drive Ø. This should be stored as the archive copy.

7. Use the disk in drive 1 as the working copy of the package or module.

II. Copy to working disk. Add the package or module to your working copy of TEK SPS BASIC on floppy disk. Depending on the number of free blocks on your working disk, you may not be able to do this.

1. Load your working copy of TEK SPS BASIC into drive Ø, bootstrap, but do not write-protect.

2. Load the original disk with the package or module into drive 1 and **write-protect** (if possible).

3. When the system is loaded and READY is printed on the terminal, type:

```
COPY DX1:"*.SPS" TO "*.SPS"
```

4. When READY is printed on the terminal, remove the original disk from drive 1. This should be stored as the archive copy.

5. Your working copy in drive Ø now includes the new package or module.

Instrument Checkout Software on Floppy Disk

Use one of these procedures to archive instrument checkout software. The type of the checkout software determines which method you use.

I. BASIC checkout software. Follow this procedure if the instrument checkout software is a BASIC program (e.g., CP56008 7912AD Checkout Software).

1. Load a blank, formatted disk into drive 1.

2. Boot TEK SPS BASIC from drive Ø. When the system is loaded and READY is printed on the terminal, type:

```
LOAD "SQUISH"
```

3. Remove the disk with TEK SPS BASIC from drive Ø.

4. Load the original disk with the checkout software into drive Ø and **write-protect** (if possible).

5. Type:

```
SQUISH DX: TO DX1:
```

6. When READY is printed on the terminal, remove the original disk from drive Ø. This should be stored as the archive copy.

7. Use the disk in drive 1 as the working copy of the instrument checkout software.

II. Stand-alone checkout software. Follow this procedure if the instrument checkout software is a stand-alone software with its own .LDA file (e.g., CP56001 P7001/R7912 Checkout Software).

1. Follow steps 1 and 2 in **I** above.

2. Enter:

```
HOOKQ DX1:
```

3. Follow steps 3 through 7 in **I** above.

When the working disk is bootstrapped, a prompt (*) will appear on the terminal. Any file with an .LDA extension can then be loaded by entering the file name, without the file name extension.

To load and execute TEK SPS BASIC, type:

```
SPSDX
```

To load and execute the instrument checkout software, type the name of the .LDA file, but without the .LDA extension.

YOUR COMMENTS COUNT

The Manual Writers at Tektronix, Inc. are interested in what you think about this manual, how you use it, and changes you might like to see in future manuals. Any queries regarding this manual will be answered personally.

What did you find that was:

interesting? _____

frustrating? _____

helpful? _____

confusing? _____

Is there anything you would like to see added to or deleted from this manual? _____

What is your major application area for this product? _____

Have you found any interesting applications, operating hints, or software routines which you would like to share with us? _____

* * * * *

Name: _____ Position: _____

Company: _____ Department: _____

Street: _____

City: _____ State: _____ Zip: _____

Fold on dotted lines and tape.
Postage will be paid by Tektronix, Inc. if mailed in U.S.A.

FIRST CLASS
PERMIT NO. 61
BEAVERTON, OREGON

BUSINESS REPLY MAIL

No postage necessary if mailed in the United States

Postage will be paid by

TEKTRONIX, INC.
P.O. Box 500
Beaverton, Oregon 97005

ATTN: SPS Documentation Group 157 – 94-384