DISKMON USER'S GUIDE

PUBLISHED BY: COMPUTHINK
965 West Maude Avenue
Sunnyvale, CA 94086
(408) 245-4033

AUGUST, 1979

TABLE OF CONTENTS

Chapter 1 — ABSTRACT

The DISKMON operating system was developed for the Commodore PET 2001 computer as a disk operating system. The DISKMON operating system adds nine additional operating system commands in the READY mode on the PET, and adds seven additional extended disk instructions to the Microsoft BASIC resident in the PET. The DISKMON operating system requires approximately 8K bytes of ROM in block B000 of PET memory.

This manual is the DISKMON User's Guide and contains much of the information necessary to supply BASIC and 6502 Assembler language programmers with technical information on the DISKMON operating system.

Chapter 2 — INTRODUCTION

Thank you for purchasing the DISKMON system. The DISKMON operating system was developed for the Commodore PET 2001 computer as a disk operating system. The DISKMON operating system adds nine additional operating system commands in the READY mode on the PET, and adds seven additional extended disk instructions to the Microsoft BASIC resident in the PET. The DISKMON operating system requires approximately 8K bytes of ROM in block B000 of PET memory.

This manual is the DISKMON User's Guide and contains much of the information necessary to supply BASIC and 6502 Assembler language programmers with technical information on the DISKMON operating system. A complete listing of the 6502 Assembler modules comprising the DISKMON operating system can be found in the manual: DISKMON ASSEMBLER LISTING.

The DISKMON operating system is supplied in ROM along with the disk system. Therefore DISKMON is always resident in the PET when it is turned on. No complicated loading procedure is required. Right after the PET is turned on the following instruction(s) will initialize the DISKMON operating system.

```
SYS(11*4096)
     or
  SYS45056
```

Either instruction need only be executed once after turning on the PET. Once initialized, the DISKMON operating system stays initialized until the PET is turned off.

The DISKMON operating system is designed to support the 6502 Assembler programmer as well as the Microsoft BASIC programmer. Machine language programs, as well as BASIC programs can be saved on disk and automatically loaded and executed. One command allows one program executing in the PET to request that it be erased and that another program be loaded into the PET and executed. Such programs may share variables or start fresh at the programmer's option. Also a BASIC program may load in a machine language program and vice versa.

The DISKMON operating system also supports any of a line of parallel printers attached to the parallel user port via Compu/Think's printer interface cable. A special extended BASIC instruction is provided in DISKMON to support this range of printers right from the BASIC program.

Diskette Insertion Procedure

After the initialization command (SYS45056), insert the diskette labeled "DEMO" into the lefthand drive (Drive number one). Notice that the diskette has a ¼" notch on one edge. The diskette should always be inserted in such a way that the notch is pointing straight up. Diskettes should be handled by the edges with great care. Before attempting any commands on the supplied demo diskettes, we suggest that you carefully read through this entire manual and study the various DISKMON commands in detail.

Chapter 3 – USER STANDARDS AND PROCEDURES

The most important aspect of the DISKMON/user relationship is that the user treat his DISKMON operating system correctly. Without this love and understanding, the DISKMON system may bite the user unexpectedly causing loss of data and a premature end to the relationship perhaps even requiring a power-off/power-on of the PET.

BEFORE STARTING

Before starting to use the DISKMON operating system, the user must purchase several diskettes. These should be 5 inch soft sectored IBM format diskettes. We recommend the DYSAN model 104 (available from dealers). The user should always purchase these diskettes in pairs. The first DISKMON standard is that every diskette must have its back-up diskette.

NEVER WRITE TO A DISKETTE WHICH DOES NOT HAVE A BACKUP COPY.

Disk errors are infrequent, especially compared to cassette tape; however, they do occur from time to time. During a write operation, a disk write error may garbage important disk data. Therefore the user *always keeps a back-up copy of each diskette and makes frequent copies from the working copy to the back-up copy* using the DISKCOPY utility program supplied with DISKMON. *Users who do not have a current back-up copy of each diskette, deserve to be bitten.* As a matter of fact, one diskette is known to have been destroyed when the user sat on it! When you buy your diskettes we suggest that you purchase a set of vinyl diskette holders. They come in ring binders and are out standard for storing diskettes for the DISKMON operating system. Each diskette and its back-up copy should be stored on the same page of the vinyl holder. These vinyl diskette holders are available from many of our dealers.

POWERING UP

When powering up the PET and its peripherals, the following procedure is recommended.

(1) ALWAYS POWER THE PET ON FIRST.
(2) ALWAYS POWER THE PRINTER ON SECOND.
(3) ALWAYS POWER THE DISK ON THIRD.

POWERING DOWN

When powering down the PET and its peripherals down, the following procedure is recommended.

(1) ALWAYS POWER THE DISK OFF FIRST.
(2) ALWAYS POWER THE PRINTER OFF SECOND.
(3) ALWAYS POWER THE PET OFF THIRD.

NOTE: Do not insert or remove any daughter boards from the slots of an expandamem while the PET's power is on.

STARTING A NEW DISK

Each time a raw (new) disk is entered into the DISKMON system, it must be formatted as the very first operation performed by DISKMON. When you purchase new diskettes we recommend that you immediately FORMAT both sides of your entire supply.

Do not save or write any program or data files on the diskette until you have read and understood these conventions. Every DISKMON user must adhere to these disk file naming conventions in order for the common disk utilities to work properly.

Simply stated a naming convention is an agreed upon method of naming things so that these things can be distinguished by name alone. The DISKMON naming conventions are designed so that file types can be distinguished both by humans and by DISKMON from the file name alone.

First of all, DISKMON disk file names can be from one to sixteen characters in length, but no longer. NEVER give a DISKMON file an all blank or null file name.

Each file is distinguished as to type by the following suffix conventions:

BASIC PROGRAM FILES	(*NONE*)
MACHINE LANGUAGE PROGRAM FILES	.GO
ASSEMBLER SOURCE FILES	.ASM
OBJECT FILES	.OBJ
FIFTH SOURCE FILES	.FIF
FORTRAN SOURCE FILES	.FOR
PLM SOURCE FILES	.PLM
PASCAL SOURCE FILES	.PAS
SEQUENTIAL DATA FILES	.DAT
LINK EDIT CONTROL STATEMENTS	.CNTL
DIRECT ACCESS FILES	.DIR

The user may further distinguish other data file types which he may develop from time to time with a . followed by any suffix of his choosing except one of the above. As an example take the following file names:

BREAKOUT
    Its name tells us it is a BASIC program

NAMES.DAT
    A sequential data file

DSKSAVE.ASM
    An Assembler source file

GRAPHICS.FOR
    A Fortran source file

MAILING.DIR
    A Direct Access file

BOLERO.MUS
    A user defined data file of category .MUS

DISK DATA FILES

The DISKMON operating system supports ASC data files and program files. Program files are BASIC program files and machine language program files (.GO). All other data files are considered ASC data files and can be read with the *$RDISK* instruction. In order for the DISKMON common utility programs to work correctly, all ASC data files have a final record usually termed "EOF." The EOF record is the last record on every ASC data file.

EOF is an acronym that stands for End Of File. DISKMON automatically writes a null character (" ") when the *$CDISK* (closedisk) command is executed. Again, this EOF character is automatically placed as the last character in any particular file. This automatic EOF allows the user to easily test (during *READ*) for the null character.

Chapter 4 — DISKMON OVERVIEW

The DISKMON operating system is designed for use with five inch mini floppy diskettes. The *$FORMAT* command of the DISKMON operating system formats each diskette for forty (40) tracks. The DISKMON operating system operates with a dual density controller. In dual density, each track contains five thousand one hundred twenty (5120) bytes of data space. A diskette formatted by DISKMON always has a disk directory located on absolute track zero. Each diskette has a total data capacity of 204,800 bytes (dual density) per side, and both sides of the diskette may be used with DISKMON.

The DISKMON operating system is designed for speed of data access. This is accomplished by reading an entire disk track at a time. It is interesting to note that, with the DISKMON operating system, the dual density controller makes data access almost twice as fast as with the single density controller The DISKMON operating system always turns the disk drive motor off after each operation to reduce diskette wear. When DISKMON recognizes an open disk command it automatically starts the disk drive motor. After .5 seconds the disk drive motor will attain the RPM speed necessary for READ/WRITE functions. *From the motor off position*, the DISKMON operating system will load a 20K PET program in less than three seconds (dual density). Similarly, a 32K program will load in less than five seconds (dual density).

The DISKMON operating system is automatically self-organizing. All free disk tracks are automatically organized and made ready for the next file allocation without any required intervention by the user. This totally eliminates time consuming disk copy operations to reorganize free tracks on a diskette.

The minimum space allocated to a file or program in the DISKMON operating system is one track. A single file may occupy multiple tracks, but no two files are ever allocated to the same physical track. Allocation of files to physical disk tracks is accomplished through the disk directory on physical track zero of each DISKMON diskette. The format for the disk directory is as follows:

SINGLE DISK DIRECTORY ENTRY

FILE NAME	(16 BYTES)
RELATIVE TRACK NUMBER	(1 BYTE)
INDEX DATA	(8 BYTES)

FIGURE 4a: Each directory entry is a total of twenty-five (25) bytes in length. The disk directory contains forty (40) such entries. The disk directory is 1,000 bytes long, and is read into locations (hex) AB00 through AEE8 in PET memory.

Any DISKMON diskette may have up to thirty-nine files allocated to it (one track is always reserved for the directory). If any file on a diskette requires more than a single track, the diskette will have less than 39 files allocated on it.

Chapter 5 — DISKMON COMMANDS

When you make the "link" to DISKMON with the command SYS11*4096 (or SYS45056), you have an additional set of commands available. Nine of these are considered direct commands and six are parts of extended BASIC. As in the "normal" commands on the PET, the distinction between direct commands and extended BASIC instructions is somewhat blurred by the fact that all BASIC commands (with very few exceptions) can be executed in direct mode and most direct commands can be used in program mode (although most of them cause the program to stop).

The nine extended direct commands are:

(1) *$FORMAT,D*
Prepares a new diskette for use.

(2) *$DIR,D*
Displays the directory information.

(3) *LOD,D,F*
Loads a program into memory from disk.

(4) *SAV,D,F,S$,E$*
Saves a program on disk.

(5) *$ERASE,E,F$*
Erases a program or file from disk.

(6) *$GO*
Begins execution of a machine language program.

(7) *$HALT*
Resets memory.

(8) *MEM,A*
Displays a page of memory.

(9) *$BLIST,F$*
Prints a listing of the current program in memory, via the user port.

The six extended BASIC instructions are:

(1) *$ODISK,D,T$,F$,I$*
Opens a datafile.

(2) *$CDISK*
Closes a datafile.

(3) *$RDISK,R$*
Reads a record.

(4) *$WDISK,R$*
Writes a data record.

(5) *XEQ,D,F*
Loads and runs another program.

(6) *$PRNT,P$*
Print a string on a parallel printer equipped with a Compu/Think cable.

We will discuss the exact details of each of these commands in the following section.

SECTION 1:
GENERAL DISK OPERATIONS
$FORMAT,D
$DIR,D
$ERASE,D,F$

SECTION 2:
MEMORY OPERATIONS
MEM,A
$HALT
$GO

SECTION 3:
PROGRAM FILE OPERATIONS
SAV,D,F,S$,E$
LOD,D,F
SXEQ,D,F$

SECTION 4:
DATAFILE OPERATIONS
$ODISK,D,T$,F$,I$
$CDISK
$RDISK,R$
$WDISK,R$

SECTION 5:
PRINTER OPERATIONS
$BLIST,F$
$PRNT,P$

NOTE: All commands may be abbreviated by using a $ followed by the first letter of the command, i.e., *$FORMAT,D* – may be abbreviated as *$F,D.* Similarly, *$SAVE,1, "PROGRAM"* – may be abbreviated as *$S,1,"PROGRAM".*

IMPORTANT NOTE:

We will be making reference to three kinds of variables and two kinds of constants.

(1) *STRING VARIABLES* are fields of ASCII-coded character data designated by a variable name ending with a dollar sign.

(2) *STRING CONSTANTS* are fields of ASCII-coded character data enclosed in quotation marks (i.e., "ANY CHARACTERS – XIY223")

(3) *INTEGER VARIABLES* are two-byte binary numbers in two's – complement format. They are designated by a variable name ending with a percentage sign (i.e., A%, AB%, AI%, A(I)%).

(4) *FLOATING-POINT VARIABLES* are numbers stored in a special format designated by the PET's system. They are designated by variable names without any special characters appended (i.e., A=5, X=6.5).

(5) *NUMERIC CONSTANTS* are numbers that are designated in a program by their numerals. The PET's operating system does not allow these to be designated as integers of the two-byte variety; i.e., A%=5 is OK and the variable A% is stored as two bytes. You cannot, however, optimize memory by designating an integer constant with a percent sign. A%-5% is therefore a syntax error.

FILE PARAMETER CONVENTIONS

DISKMON allows you to use numeric constants or any floating-point variable names for numeric parameters (the drive number and record number). It also allows you to use any string constant or string variable for other parameters. This allows you a great deal of freedom, but can lead to confusion. To help eliminate this potential confusion, we suggest conventional variables for each of these parameters:

Disk file name —	F$
Disk file record —	R$
Drive number —	D
Type access code ("W", "R", or "D") —	T$

Note: The Type access Code ("W", "R", "D") must *always* be enclosed in quotation marks.

Index data string —	I$
Record number —	N
Starting address —	S$
Ending address —	E$

We recommend that you simplify your program debugging problems by using these variables in all your programs.

Chapter 6 — SECTION 1: GENERAL DISK OPERATIONS

$FORMAT,D

THIS IS A CRITICAL COMMAND!!!

It prepares a new diskette (one side at a time) for use. Every diskette that you use on this system must be formatted with this command BEFORE it is used with any other command. There are a wide variety of unpredictable antics that your equipment will perform if you attempt to use a diskette that has not been formatted. The best way to avoid any of these errors is to prevent any unformatted diskettes from finding their way into your supply. When you get a new supply of diskettes, just sit down and format both sides of all of them before placing them in your supply of available blanks. If you don't have any unformatted diskettes around, you can't have any of these errors.

The parameter D in the example above can be replaced by any valid numeric constant or floating-point variable. It designates the drive on which the formatting operation is to be performed. As a convention however, we suggest that you use the varible D.

EXAMPLES:

```
$FORMAT,D
$FORMAT,1
```

Both perform the same if D=1.

This command can also be used to erase (re-format) one side of a diskette. This enables the diskette to be reused.

$DIR,D

This command displays the names of the programs and files stored on the diskette designated by the numeric constant or floating-point variable D (in the example above). In all of our examples D represents a variable which designates the drive on which the operation is to be performed.

$ERASE,D,F$

This command erases the program or file from the diskette and removes the directory entry for the file designated by F$. The file to be erased is designated by a string variable (F$ in our example above) or a string constant. Again, the variable D represents the drive number (1 or 2).

EXAMPLES:

```
$ERASE,A,"FILENAME"
$ERASE,A,F$ (Where "FILENAME" has already been assigned to F$)
$ERASE,1,"FILENAME"
$ERASE,1,F$
```

These commands will all perform identicaly if D=1 and F$="FILENAME". The program named "FILENAME" will be erased from the diskette on drive #1.

SECTION 2: MEMORY OPERATIONS

MEM,A

This command displays a section of the PET's memory on the screen. The portion of memory is designated by an address in four-digit hexadecimal format. The hexadecimal number is assigned a string variable or constant value (A$ in the example above).

EXAMPLES:

```
$MEM,"B000"
$MEM,A$
```

Both perform the same if A$="B000". A portion of memory beginning at address perform 45056 ($B000) will be displayed.

$HALT

This command clears the computer's memory almost as though you had turned it off and back again. The "link" to DISKMON however, is preserved when this command is executed.

$GO

When a machine-language program is loaded into memory, a normal "RUN" command will not begin execution of that program (the pointers are handled differently from the way BASIC program pointers are set). This command will begin execution of the machine language program.

SECTION 3: PROGRAM FILE OPERATIONS

SAV,D,F,S$,E$

This command actually has two formats — one for machine-language programs and one for BASIC programs. The format above is used for machine-language programs. This command designates a block of memory with two hexadecimal numbers assigned to strings — either a string variable or a string constant. In our example above, the starting address of the machine-language program is designated by S$ and the ending address is designated by E$. The hexadecimal format is exactly the same as for the *MEM,A* command. The block of memory stored in this manner is given the name F$ (any string variable you wish to use — or a string constant) and stored on the diskette in drive D.

EXAMPLES:

SAV,D,F,S$,E$
$SAV,1,"FLASH.GO","0F00","0FF0"

Both perform the same if D=1, F$="FLASH.GO", S$="0F00", and E$="0FF0". The machine-language program (already in memory) starting at address 3840 ($0F00) and ending at address 4080 ($0FF0) is stored on the diskette in drive 1 under the name "FLASH.GO". When this program is loaded back into memory, it will automatically be located at the same memory addresses (overlaying the same block of addresses). Ending address will always be a multiple of 512 bytes or 1 sector, and not necessarily what is specified.

This command may be used to save ANY block of data from memory. Creative programmers might use it to store a large block of variables or various sections of a program to use as overlaid subroutines.

SAV,D,F

This is the format used to save BASIC programs. It is exactly the same as the format for machine-language programs except that the starting and ending addresses are not specified. All other parameters are identical.

LOD,D,F

This command is the reverse operation of the *SAV,D,F* command. The parameters D and F$ still refer to the drive and file name, respectively. This command may also be used to load a machine-language program — which will then be executable by a $GO command. BASIC programs, of course, may begin execution with a RUN command. Machine-language programs, when loaded into memory, will automatically be loaded into the same block of memory as was specified by the S$ and E$ addresses, when the program was saved on diskette.

A very important alternate format is:

$LOD;D,F$

This version of the command performs the same loading actions, but does not reset any of the program pointers. It is extremely useful for overlaying a variety of machine-language subroutines from a single BASIC program. BE VERY CAREFUL when using this technique. The machine-language subroutines loaded in this manner cannot be allowed to interfere with the BASIC program. To utilize this command requires that the user be somewhat familiar with assembly language programming.

Since the only syntax difference between the different forms of the *LOD,D,F* and *$LOD;D,F$* is the use of either a comma or a semicolon, you must exercise great care whenever you use this command in a program. Mistakes in direct mode tend to be less costly and frustrating.

XEQ,D,F

This command performs two functions — it first loads into memory the program designated by F$ from the diskette in the drive designated by D, and then it automatically begins execution of that program. For a BASIC program, this is equivalent to the combination of commands:

LOD,D,F:RUN

and for a machine-language program it is equivalent to:

LOD,D,F:GO

This command also has two formats -- again with the only distinction being the use of either a comma or a semicolon. The two different functions are usually described as CHAINING to another program or OVERLAYING another program. The format described above represents the chaining option. The primary disadvantage is that all the variables and pointers of the first program are lost when the new program is brought into memory. The advantage is that you do not have to pay attention to memory allocation. A small BASIC program can chain to a larger BASIC program with no problem.

The alternate format is:

$XEQ;D,F$

This command loads the program designated by F$ into memory from the diskette on drive D and preserves the variables and other data (system pointers) in memory. It overlays only the program area of memory. There is a user disadvantage to overlaying BASIC programs in this manner. The PET's operating system allocates as much memory as necessary for the program and begins allocating memory to variables immediately after the end of the program. If the program loaded by this command is larger than the program which created the variable values, the variable values will be lost. In some instances you can cause your system to "crash" by the improper allocation of memory. The only foolproof rule is to be sure that the first program in a group to be overlaid on each other is the biggest one in the group. When you check the program sizes, check the number of bytes free BEFORE the program is run. This will tell you how many bytes are free before any memory is allocated to variables. Second loaded program must always be 512 bytes SMALLER than first or one sector smaller. There is another peculiarity involving the PET's treatment of string data that we will discuss again in a special section ("string variable anomolies") at the end of this chapter.

SECTION 4: DATAFILE OPERATIONS

Datafile operations are a bit more complex than any of the other operations and an adequate explanation of the datafile commands requires subsections.

COMMAND PARAMETERS

$ODISK,D,T$,F$,I$

This opens a datafile on drive D for a type of access designated by T$. The name of the file opened is designated by F$ and the index data is indicated by I$. These terms will be fully explained later.

$CDISK

This closes the datafile currently opened. If the file was opened for WRITE then this command automatically writes a null character as an EOF.

$RDISK,R$

This reads the next sequential record into the variable R$.

$WDISK,R$

This writes the contents of the variable R$ to the next record of the file.

$RDISK;N,R$

This reads the contents of record number N from a direct-access file into the variable R$.

$WDISK;N,R$

This writes the contents of R$ into record number N in a direct-access file.

FILE PARAMETERS

Datafiles contain only ASCII characters (string data). Numeric data is normally stored by the proper numerals (using the STR$ and VAL functions for conversion). Numeric data can be stored in a more compacted form by using the CHR$ and ASC functions for conversion. Unless there is a recordsize problem, this is not usually worth the extra effort.

Each record has a carriage return CHR$(13) as the terminator. A record 30 characters long will actually occupy 31 bytes on the diskette. This may be important to someone who really wants to allocate the 5120-byte tracks extremely efficiently.

TYPES OF ACCESS

There are three types of file access in this system — sequential write (code W), sequential read (code R), and direct access (code D).

I. SEQUENTIAL WRITE ACCESS

This is the only method by which a new diskette file can be created. The access code (T$ is our usual example) is "W", which represents WRITE.

$ODISK,D,T$,F$,I$
$ODISK,1,"W";"MAILING LIST";"12/31/78"

Both perform the same if D=1, T$="W", F$="MAILING LIST", and I$=12/31/78. A datafile named "MAILING LIST" is created on drive 1 in preparation for records to be written to it. The directory alos contains an eight-character string of index data. If the file is to be accessed by a direct-access method at some time in the future, the index data must have a special structure that we will explain later. In our example, here, the file will only be accessed sequentially. The index data therefore, can be used to store any pertinent information. In this example, the index data is used to show how current the data is:

```
$WDISK,R$
$WDISK,"HENRY JOHNSON"
```

This writes the contents of R$ (or a string) to the next record in the file.

```
$CDISK
```

This closes the file. In the WRITE mode it also automatically writes a null (" ") record as an end-of-file mark. Be sure that you do not have any other null records in your files — null records are reserved for the end-of-file marker.

II. SEQUENTIAL READ ACCESS

This is the only mode in which files not structured for direct access can be read. The access code is "R" which represents READ.

```
$ODISK,D,T$,F$,I$
$ODISK,1,"R","MAILING LIST",I$
```

Notice the important difference in this pair of equivalent examples — the index variable (I$ is our example) *CANNOT BE REPLACED BY A STRING CONSTANT.* The command will open the file named "MAILING LIST" on drive 1 for read access and place the index data into the variable I$. The method of placing the data into I$ leads to a possible anomoly (the same one mentioned in the section on program overlays) that is discussed at the end of the chapter.

There is an important feature of error trapping built into the system. If the file you attempt to open is not on the designated diskette, the system sets location 44976 to a value of 255. That's $AFB0 in hexadecimal format. This allows you to set up your own error-handling routines for a "file not found" error.

```
$RDISK,R$
```

There is no alternate version. The next record in the file is read into the designated variable. As in all these command examples, any variable name can be used. For convention purposes, we suggest that the string variable R$ be used to represent a record. The possible string anomoly (see separate section on Anomolies) can occur on this operation.

```
$CDISK
```

This closes the file, but when READING a file does not write an end-of-file mark. You may close the file after reading only part of it without destroying any of the remaining data.

III. DIRECT (UPDATE) ACCESS

This access mode is very powerful, but the file must be highly structured and the potential for error requires extra caution. Back-up data is always important, but in this type of file, back-up data becomes a critical necessity.

The access code for this mode is "D" and all files must have a defined number of records that ALL are of a defined length. This information is stored in the first four characters in the index data. (Index data is 8 characters long.)

Index data structure:

The first two characters in the index data are actually a compacted integer in the reverse format common to 6502-based equipment. The integer contains the number of records in the file. The next two characters are in the same format and define the length of the records in the file. If you, in error, try to write a record of the wrong length to the file, DISKMON will respond with a SYNTAX ERROR and stop execution of the program. The last four characters of the index can be used in any way you wish. As long as these restrictions are met, however, you will be able to do the following:

Extract any record from any position in the file, make any changes you wish to record, and place it back in the file — all without disturbing the other records. That potential makes all the effort that goes into maintaining this rigid file structure worthwhile.

See the "Creating a Random Access File" or RANDOM-FORMAT sample programs for examples of forming the proper index data. We suggest that you use the RANDOM-FORMAT utility program to format your Direct Access Files.

The open and close commands are identical to those used for read access, so we won't repeat them. The read and write commands are different, however.

$RDISK;N,R$

This reads the Nth record in the file into the variable R$. Other than the designation of a particular record by number, this command works exactly like a sequential read. N can be any numeric constant or valid floating-point variable within the range of record numbers.

$WDISK;N,R$

This reads the contents of R$ in memory and writes it to the Nth record of the file.

Notice the use of the semicolons in these commands. They serve as notice to the operating system that this is a direct operation. Be careful that you do not substitute commas.

Various utility programs (and their respective listings) to format, read, and update a Direct Access File have been provided.

SECTION 5: PRINTER OPERATIONS

The two commands covered in this section describe the commercial printer support available by connecting the proper cables (available from Compu/Think and various dealers) from the proper type printer to the parallel port of your computer. If you have some other type of printer (probably connected to the IEEE-488 port) these commands will not apply to your system.

$BLIST,"F$"

This causes the BASIC program currently in memory to be listed on the printer. The program name ("F$" is our example above) may be a string variable, but will usually be given as a string constant and must be enclosed in quotes. It is used solely to print a program heading. This command will not cause the system to load the named program into memory before listing it.

When the program is listed, it will be listed on individual pages (50 lines per page) with a heading and page number at the top of each page. There are two assumptions made:

1) That there is a BASIC program in memory.
2) That you have positioned the printer to the top of a new page.

$PRNT,XGT$;Y(I),R%,A:"EXAMPLE"

This functions just like the normal PRINT command except that the resulting characters are put on the printer paper rather than on the screen. Commas and semicolons have the same effect. Both characters will automatically "pack" the characters together, i.e., *$PRNT A,B* will print on the printer identically to *$PRNT A;B*.

Chapter 7 — STRING ANOMOLIES

The PET's operating system handles string data with tables that hold the variable name, the current length of the string, and the location where it actually begins. If you set a string variable equal to a string constant in a BASIC program, the operating system will set the pointer for that string variable to the actual part of the program itself where the constant begins. This is an efficient way to save memory, but it causes strange things to happen when you overlay that program with another. The pointer remains the same but the data it points at is changed — creating various unpleasant surprises.

A similar thing happens when a record is read from a disk file. DISKMON brings the records into a buffer area and sets the pointer of the variable you have designated so that it points at this buffer. The same thing is done for index data when a disk datafile is opened for sequential read or direct update operations.

If you read several different variables from a disk datafile, they will all point to the same place, so they will all be equal to each other. One example:

```
$RDISK,R$
$RDISK,Z$
```

The variable pointers for both R$ and Z$ will now point to the DISKMON buffer area. The original data for R$ is lost and the two are equal to each other.

There is a cure for all of these problems, however. A peculiar-looking, but entirely valid operation setting a string variable equal to itself will cause the PET's operating system to move the data into the string holding area and reset the variable pointer. Our example above would change to:

```
$RDISK,R$
R$=R$
$RDISK,Z$
```

Now R$ will be held in the PET's normal string area and Z$ will be in the DISKMON buffer.

In the same way, if your last operation on a string variable in a program was to set it equal to a constant, you must set it equal to itself before bringing in an overlay program that would destroy the values. The PET would recopy that constant data into the string holding area and reset the pointer.

PROGRAM ANOMOLIES

There are a variety of rather esoteric "bugs" in the PET's operating system that may cause problems. Some anomolies can occur when DISKMON is "linked" to the PET's operating system.

1) Screen editor changes:

In long program lines (usually 77 characters or more) after the screen edit functions, there are sometimes problems with "scrambled" characters appearing at the end of the line. In some other cases, you might make a small change on a line that does not "take" (when you list the line again, the change was not made). Some other types of errors also occur when the screen editor is used with DISKMON in residence.

If you encounter any of these problems, there is a rather simple cure — just restructure the long program line and divide it into two or more shorter lines.

2) DISKMON commands as part of an IF statement sometimes execute when th should not. The cure for this is a rather peculiar-looking, but valid command with a colon follow ing the THEN part of the IF . . . THEN statement.

EXAMPLE:

```
50 IF PEEK(44976) <> 255THEN$CDISK:GO TO 90
```

This line will often close the disk no matter what the value of PEEK(44976) may be. If we change it to:

```
50 IF PEEK(44976) <> 255THEN:$CDISK:GO TO 90
```

Now, it will work the way it should. This little peculiarity is due to the way DISKMON interacts with the BASIC commands in the PET. As long as there is a colon in front of the first DISKMON extended BASIC command in the conditional part of the IF statement, the command should execute properly.

PROGRAMMING HINTS

Error Trapping

There is a potential problem with a file open command if the file does not exist. In direct mode (i.e., *$LOAD,2,F$*), a "File not found" error message will be displayed.

In program mode, however, DISKMON will set decimal location 44976 to 255. This allows you to "trap" the error.

To allow you to write your own routines, DISKMON will not go into its standard error routine. DISKMON sets decimal location 44976 to a value of 255 which can be tested under program control Using the "flag" properly requires an opening routine such as the one below:

```
50 POKE44976,0:$0DISK,1,"R";F$
55 REM CLEAR ERROR FLAG, OPEN FILE,
60 IF PEEK(44976)=255 GO TO 1000
65 REM CHECK ERROR FLAG
1000 PRINT"NO SUCH FILE ON DRIVE";D
1010 INPUT"NEW FILE NAME";F$:GO TO 50
```

Recommendations

When in direct mode we recommend that the DISKMON user execute the *$HALT* command between progran LOADs. The *$HALT* command efficiently refreshes/erases RAM and reduces the possibility of value anomolies occuring.

As you insert a diskette, a microswitch will enable (you hear a click) the WRITE PROTECT NOTCH on the diskette. This will occur during the last 3-5 millimeters of insertion. When inserting a diskette (with the notch up) always carefully insert the diskette until you hear this microswitch enable. At this point the diskette will be properly seated. If you have purchased a set of diskettes with your system you will also receive "write protect" tabs which fold over, and protect, the notch. These tabs will prevent the microswitch from enabling which makes it impossible to inadvertently write to a diskette. When attempting to write to a protected diskette, DISKMON will respond with a DISK FULL ERROR

PRELIMINARY TROUBLESHOOTING

**ALWAYS CHECK YOUR INSTALLATION PROCEDURES*

Directory Frame Without Contents:

If drive motor is not on check disk power on switch: a red LED power light should activate on the front panel. Remove diskette and re-insert. If problem continues, power down system, power back up and try in this sequence: Use a new (or reusable) diskette, FORMAT the diskette, and call for a Directory. DISKMON should display a directory frame, and within one second display 39FREE TRACKS. If problem continues, power down, check all components, and gently move the two daughter boards (DCC and DOS) into different slots on the EXPANDAMEM. Try procedures again. If problem persists remove disk cover (remove the cover vertically) and check all ribbon cable/plug connectors. If problem continues, contact your dealer.

Both Drives Run Simultaneously:

Check the Disk Controller Card ribbon cable to the drive unit. The cable should have a half-twist from the DCC to the drive unit. Blue wire edge should be on top where the cable plugs into the drive unit. If problem persists, remove disk cover and check the DIP switches on the left side of each individual drive. The left drive (No. 1) should have the DIP switch No. 1 enabled. Drive No. 2 should have DIP switch No. 2 enabled.

Noisy Wobbly Diskette:

Remove diskette and check the center spindle hole. The diskette can 'move' a few millimeters within the diskette jacket. If edge of spindle hold is damaged, this indicates improper insertion. If problem continues check spindle inside drive.

String Value Anomolies:

To reduce the possibility of string value anomolies we suggest that you use the $HALT between all LOADS while in direct mode. Where sharing variable values between programs is not important insert the CLR command in the first line of all programs.

Disk Error Messages:

The DISKMON operating system uses one error message for all disk drive errors. This message indicates a malfunction on the disk drive and may be interpreted as follows:

DISK ERROR E=ee, T=tt, S=ss

E=error number (as follows below), T=track number, and S=sector number.

ErrorNumbers:	00=verify error	2X=write-unrecoverable error	4X=write protected
	8X=not ready	X8=read-unrecoverable error	X4=lost data
		1X=disk has not been formatted	

*X may be any hexidecimal digit

NOTE: These error numbers are the hexidecimal value of the status byte returned by the Western Digital FD1791A Floppy Disk Controller.

Chapter 8 – DISKMON UTILITY PROGRAMS

The DISKMON operating system is supplied with two diskettes. One is a blank diskette, and the other is a demonstration diskette on which reside a number of utility programs which perform important and useful tasks for the user. Programs are located on both sides of the demonstration diskette. These programs are:

MONITOR
DATAFILE
DISKTEST
DISKCOPY
RANDOM-FORMAT
PAGETEST
BLOCKTEST
DENSITYCOPY

MONITOR

Monitor is a program which allows the user to interact with the PET at the machine language level. Monitor contains a memory dump routine, a 6502 disassembler, a memory move routine, a program testing routine, and a tiny 6502 Assembler which uses the same mneumonics as the larger PET-ASSEMBLER.

DATAFILE

Datafile is a utility for handling ASC data files only. Datafile allows the user to copy, display, or print the contents of an ASC data file. Datafile should not be used with program files. It is for data files only. In the copy mode, the Datafile program will prompt the user for the FROM file and the TO file. If the TO file already exists, the FROM file will be appended to the end of the TO file instead of copied over the TO file.

DISKTEST

The Disktest program allows the user to test any of the drives in his system. Drives are tested one at a time. A blank diskette is required for testing. A test data file is written on the diskette during testing.

DISKCOPY

The Diskcopy program allows the user to copy the entire contents of a diskette onto another diskette. The Diskcopy program requires two drives and two diskettes. The program allows the user to copy from drive one to drive two or from drive two to drive one. The Diskcopy program automatically alters itself to reflect the amount of available memory present in the PET.

RANDOM-FORMAT

The Random-format program allows the user to format an ASC data file for later random access. Random access files must be specially formatted before using. The program allows the user to specify the file name, the disk number, the number of records, and the record length. If a file already exists by that name, the Random-format program will erase the old file, and place the new random access file in its place.

BLOCKTEST & PAGETEST

Blocktest and Pagetest are memory test routines for the PET. These two routines are designed to work with the PET which has a DISKMON operating system. Each test routine works essentially the same, except one tests memory by 4096 byte blocks, and the other tests memory by 256 byte pages.

DENSITYCOPY

The DENSITYCOPY program allows you to convert a single density DISKMON diskette into dual density format. This is of value to those customers who purchased Compu/Think's original single density disk drives.

Chapter 9 — MISCELLANEA

Good software is an important component of any computer system. One of the reasons why we value good software so highly is because it takes so much time and effort to produce a really quality piece of software. In the past, we have not done enough to attract the kind of talent which the microcomputer industry needs in order to provide good consumer software. In fact, when talent has shown itself, we have quite often abused it by illegal copying and pilferage.

The DISKMON operating system contains an attempt to combine the forces of both the hardware and software in order to provide a secure environment for consumer software. Two services have been provided. First, locations $BFF0 through $BFF5 contain six ASC digits which provide a unique serial-number for each DISKMON system sold. Second, the DISKMON operating system is constructed to lock out all user instructions from READY mode (except *$HALT*, *$GO*, and *RUN*) when location 1034 is set to decimal 42.

DISKMON SPECIAL MEMORY LOCATIONS

START-END	DESCRIPTION
0006	SECURITY SWITCH ($64=SECURITY ON)
9000-A3FF	DISKMON TRACK I/O BUFFER AREA (2560 BYTES)
AA00-AAFF	DISKMON RECORD I/O AREA FOR DATA FILES (DR$)
AB00-AEE8	DISKMON DISK DIRECTORY AREA (1000 BYTES)
AF00-AF0F	DISKMON HOLDING AREA FOR DF$ VARIABLE
AF10-AF17	DISKMON HOLDING AREA FOR DI$ VARIABLE
AFD0-AFD1	DISKMON BLOCK BYTE COUNTER
AFD2	DISKMON I/O SWITCH (1=READ, 2=WRITE)
AFD3	DISKMON RECORD BYTE COUNT (DR$)
AFE0-AFEF	DISKMON CURRENTLY OPENED FILE NAME
AFF0	DISKMON CURRENT RELATIVE TRACK NUMBER
AFF1-AFF8	DISKMON CURRENT FILE INDEX DATA
AFF9	DISKMON DIRECTORY WRITE SWITCH ($10=WRITE)
AFFA	COMMAND MODE SWITCH ($22=COMMAND MODE)
AFFB	CURRENT DISK DEVICE # IN ASC ($31 or $32)
AFFC	ABSOLUTE SECTOR NUMBER
AFFD	CURRENT DISK COMMAND BYTE
AFFE	REGISTER SAVE AREA
AFFF	CURRENT ABSOLUTE DISK TRACK NUMBER
B000-B041	DISKMON ASSEMBLER PROGRAMMER'S JUMP TABLE
BFF0-BFF5	DISKMON UNIQUE DEVICE IDENTIFIER (OLD PET)
8FF0-8FF5	DISKMON UNIQUE DEVICE IDENTIFIER (NEW PET)

ASSEMBLER PROGRAMMER'S JUMP TABLE

STARTING ADDRESS = B000

B000	DISKMON INITIALIZATION ROUTINE
B003	DISKMON RETURN ADDRESS FOR EXTENDED BASIC COMMANDS
B006	DISKMON OPEN DISK DRIVE AND READ DIRECTORY RTN
B009	DISKMON CLOSE DISK DRIVE AND WRITE DIRECTORY RTN
B00C	DISKMON CLEAR CURRENT FILE DATA AREA
B00F	DISKMON ALLOCATE FILE TRACK ROUTINE
B012	DISKMON ALLOCATE FILE TRACK ROUTINE
B015	DISKMON ERASE FILE ROUTINE
B018	DISKMON WRITE DISK BUFFER ROUTINE
B01B	DISKMON READ DISK BUFFER ROUTINE
B01E	DISKMON WRITE TRACK FROM MEMORY ROUTINE
B021	DISKMON READ TRACK INTO MEMORY ROUTINE
B024	DISKMON TURN DISK MOTOR ON ROUTINE
B027	DISKMON TURN DISK MOTOR OFF ROUTINE
B02A	DISKMON SAVE PROGRAM FILE ROUTINE
B02D	DISKMON PRINT LINE ON PRINTER ROUTINE
B030	DISKMON LOAD PROGRAM FILE ROUTINE
B033	DISKMON READ FIRST TRACK OF PROGRAM FILE RTN
B036	DISKMON READ NEXT TRACK OF PROGRAM FILE RTN
B039	DISKMON ASC/HEX CONVERSION ROUTINE
B03C	DISKMON XEQ/XER LOAD AND GO ROUTINE
B03F	DISKMON SAVE RELOCATABLE PROGRAM ROUTINE

```
*************
RANDOM-FORMAT
*************


BASIC LISTING          PAGE 01


100 REM THIS PROGRAM FORMATS AN ASC
110 REM DATA FILE FOR RANDOM ACCESS
120 REM ALL RECORDS IN A RANDOM
130 REM ACCESS DATA FILE MUST BE OF
140 REM THE SAME LENGTH.  THE FILE
150 REM INDEX MUST BE SET TO STORE
160 REM THE NUMBER OF RECORDS AND
170 REM THE LENGTH OF EACH RECORD
200 REM FORMAT SCREEN AND INPUT
210 PRINT"&&RANDOM-FORMAT&&"
220 INPUT"ENTER FILE NAME ";DF$
230 INPUT"ENTER DRIVE # ";D
240 INPUT"NUMBER OF RECORDS ";RN
250 INPUT"RECORD LENGTH ";RL
260 PRINT"CREATING FILE"
270 R$=""
280 FORI=1TORL:R$=R$+" ":NEXTI
300 REM CREATE RANDOM ACCESS FILE
310 A=RN:GOSUB400
315 DI$=CHR$(L)+CHR$(H)
320 A=RL:GOSUB400
325 DI$=DI$+CHR$(L)+CHR$(H)+"     "
330 $ODISK,D,"W",DF$,DI$
340 FORI=1TORN
350 $WDISK,R$
360 NEXTI
380 $CDISK
390 END
400 REM CONVERT TO INTEGER BYTES
410 H=INT(A/256)
420 L=A-(H*256)
430 RETURN
```

```
*********
PAGETEST
*********


BASIC LISTING            PAGE 01


5 PRINT""
10 PRINT" MEMORY TEST PROGRAM:PSEUDO RANDOM DATA"
20 REM BY RICHARD TOBEY COPYRIGHT 6/16/1978
30 PRINT"":PRINT"   FOR FURTHER DESCRIPTION RUN 1000":PRINT""
40 GOSUB 700
100 INPUT"FROM PAGE";A:INPUT"TO PAGE";B
120 POKE 178,0:POKE 179,A:POKE 180,B+1
130 INPUT"# PASSES,0-255,0=CONTINUOUS";P
140 P=INT(P):IF P>255 GOTO 130
150 IF P>0 THEN C=0:GOTO 170
160 C=1:P=255:REM NO PASSES=CONT
170 INPUT"DELAY SECONDS";D
180 Q=0:R=P
190 POKE 181,0
200 SYS(832):REM START FILL
210 IF D=0 GOTO 250
220 Y=D :REM DELAY
230 FOR X=1 TO 930:NEXT
240 Y=Y-1:IF Y<>0 GOTO 230
250 SYS(848):REM START TEST
260 IF PEEK(187)=0 GOTO 360 :REM ERROR TEST
265 PRINT""
270 PRINT"ERROR IN BLOCK";INT(PEEK(183)/16)+1
275 PRINT"( PAGE";PEEK(183);")","AT LINE";PEEK(182)
280 PRINT"":PRINT"TEST DATA     ";PEEK(185),"MEMORY ";PEEK(186):PRINT""
290 PRINT"NEXT LOCATION ?"
300 GET C$:IFC$="" GOTO 300
310 IF C$="Y" THEN SYS(908):GOTO 260
320 PRINT"":PRINT"RETEST?"
330 GET C$:IF C$="" GOTO 330
340 IF C$="Y" GOTO 180
350 END
360 Q=Q+1:PRINT"PASS";Q,"TIME  ";TI$
370 R=R-1:IF R<>0 GOTO 190
380 IF C=1 GOTO 400:REM CONT FLAG SET
390 PRINT"TEST COMPLETE":GOTO 320
400 P=255:GOTO 180
700 REM SUBROUTINE TO LOAD MACHINE CODE
710 REM INTO THE SECOND CASSETTE BUFFER
720 FOR I=832 TO 912:READ W:POKE I,W:NEXT
730 RETURN
740 DATA 160,0,32,102,3,32,111,3
750 DATA 145,182,32,118,3,208,246,96
760 DATA 160,0,32,102,3,32,111,3
770 DATA 209,182,208,37,32,118,3,208
780 DATA 244,169,0,133,187,96,165,178
790 DATA 133,182,165,179,133,183,96,165
800 DATA 182,69,183,69,181,96,230,182
810 DATA 208,6,230,69,165,69,197,66
820 DATA 96,133,71,177,68,133,72,169
```

```
********
PAGETEST
********


BASIC LISTING          PAGE 02


830 DATA 255,133,73,96,160,0,76,92,3
1000 PRINT"":PRINT"THIS ROUTINE FILLS THE MEMORY WITH A
1005 PRINT"DIFFERENT PSEUDO RANDOM PATTERN
1010 PRINT"EACH PASS,THEN AFTER A DELAY PERIOD,
1020 PRINT"GENERATES THE PATTERN AGAIN AND
1030 PRINT"COMPARES IT TO THE CONTENTS OF MEMORY,
1040 PRINT"THUS CHECKING FOR POSSIBLE FAILURE
1050 PRINT"MODES,INCLUDING BAD CHIPS,STUCK
1060 PRINT"ADDRESSES,INADEQUATE REFRESH ETC.
1070 PRINT"":PRINT"  TO RUN THE PROGRAM
1080 PRINT"BLOCKS ARE 4K BYTES.  TO TEST THE 3RD
1090 PRINT"THRU 6TH BLOCKS OF MEMORY(16K ADDED)
1100 PRINT"USING SELECTS 2 THRU 5,(DEC 8192-24576)
1110 PRINT"FROM BLOCK IS 3,WHILE TO BLOCK IS 6.
1120 PRINT"THE # OF PASSES MAY BE FROM 1 TO 255
1130 PRINT"ENTERING 0 PASSES GIVES CONTINUOUS
1140 PRINT"TESTING
1150 PRINT"ALSO DELAY MAY BE ENTERED IN SECONDS
1160 PRINT"0=NO DELAY
```

```
********
MONITOR
********


BASIC LISTING          PAGE 01


100 REM INITIALIZE VARIABLES
101 IFLX<>0THEN7230
102 POKE53,48:POKE51,48;POKE49,48;MO$="3000"
105 PRINT"INITIALIZING":DEF FNA(A)=9*INT((A-48)/16)+A-INT(A/16)*16
135 P$="0393A4B5B1B691B888D0F9B1B691B860":GOSUB7100
137 P$="037AAD9003AE9103AC92032000008D90038E91038C92036000000":GOSUB7100
140 Q$=CHR$(34)
145 DIM O$(15)
155 GOSUB2200
199 GOTO6200
900 REM CONVERT OCTAL NUMBER
910 A=0:XK=LEN(A$)
920 FORIK=XKTO1STEP-1
930 A=A+(VAL(MID$(A$,IK,1))*(8^(XK-IK)))
940 NEXTIK:RETURN
950 B=0:XA=A:FORIK=5TO0STEP-1
960 B=B+(INT(XA/(8^IK))*(10^IK))
970 XA=XA-(INT(XA/(8^IK))*(8^IK))
980 NEXTIK:RETURN
1500 REM TINY PET-ASSEMBLER
1505 PRINT"&&TINY PET-ASSEMBLER&&"
1515 INPUT"LOAD ADDR";A$:GOSUB6695:LA=A
1516 FORV=1TO20
1517 PRINT"PLACE QUOTES BEFORE EACH INPUT LINE"
1518 PRINT"PLACE QUOTES BEFORE EACH INPUT LINE"
1519 NEXTV
1520 OA=LA
1530 PRINT"LOAD  ORG    OPC OPERANDS  ASSEM OUTPUT"
1535 FORMK=1TO20
1540 T=LA:GOSUB6640:PRINT"  ";
1545 T=OA:GOSUB6640
1550 INPUTS$:S$=S$+BB$
1555 IFLEFT$(S$,3)="END"THEN1599
1560 GOSUB3100
1565 IFER$<>""THENPRINT"";TAB(27);ER$:GOTO1590
1570 PRINT"";TAB(27);LEFT$(P$+BB$,12)
1575 S=LA:GOSUB7900
1580 L=LEN(P$)/2
1585 OA=OA+L:LA=LA+L
1590 NEXTMK
1599 GOTO6320
2200 REM INITIALIZE ALL VARIABLES
2205 MT$="ADCANDASLBITCMPCPXCPYDECEORINCJSRLDALDXLDY"
2206 MT$=MT$+"LSRORAROLRORSBCSTASTXSTYJMP"
2207 MT$=MT$+"BCCBCSBEQBMIBNEBPLBVCBVSBRKCLCCLDCLICLVDEXDEYINXINYNOPPHAPHPPLA"
2208 MT$=MT$+"PLPRTIRTSSECSEDSEITAXTAYTYATSXTXATXSORGEXTBYTADRTXTEND"
2210 JT$="111118811191881111111122222222233333333333333333333333456667"
2215 OP$="61210620C1E0C0C241E220A1A2A04201262E18182804090B0F030D01050700018"
2216 OP$=OP$+"D858B8CA88E8C8EA4808682840603 8F878AAA898BA8A9A000000010200"
2220 AM$="FEFE3D14FE16163CFE3C10FE5E3E3DFE3D3DFEFE1C1C10"
```

```
*******
MONITOR
*******


BASIC LISTING           PAGE 02


2230 H$="0123456789ABCDEF"
2250 BB$="                 "
2255 CC$=""
2260 RETURN
2800 REM CONVERT DECIMAL TO HEX STRING
2810 J8=2:A$=""
2820 FORI8=1TOJ8:K8=J8-I8:E=16^K8
2830 A$=A$+MID$(H$,INT(A/E)+1,1)
2835 A=A-INT(A/E)*E
2840 NEXTI8:RETURN
2850 J8=4:A$="":GOTO2820
2900 REM CONVERT HEX STRINGS TO DECIMAL
2910 A$=RIGHT$("0000"+A$,4):A=0
2920 FORI8=1TO4:A=A+16^(I8-1)*FNA(ASC(MID$(A$,5-I8,1))):NEXTI8:RETURN
3000 REM PARSE ADDRESS OPERANDS
3002 XQ=ABS(4*(MID$(JT$,N,1)="8"))
3005 C=1:O=8:IFMID$(L$,1,2)="A "THENOP=OPAND251:GOTO3092
3010 S=2:C=2:O=ABS(8*(MID$(JT$,N,1)="1")):IFMID$(L$,1,1)="#"THEN3098
3015 S=2:IFMID$(L$,1,1)<>"$"THEN3050
3020 C=4:O=4:IFMID$(L$,4,1)=" "THEN3098
3025 C=8:O=20:IFMID$(L$,4,2)=",Y"THEN3098
3030 IFMID$(L$,4,2)=",X"THEN3098
3035 C=16:O=ABS(12*(MID$(JT$,N,1)<>"9")):IFMID$(L$,6,1)=" "THEN3094
3040 C=32:O=28:IFMID$(L$,6,2)=",X"THEN3094
3045 C=64:O=24+XQ:IFMID$(L$,6,2)=",Y"THEN3094
3047 GOTO3090
3050 S=3:IFMID$(L$,1,2)<>"($"THEN3090
3055 C=16:O=44:IFMID$(L$,5,1)=")"ANDOP=64THEN3098
3060 C=128:O=0:IFMID$(L$,5,3)=",X)"THEN3098
3065 O=16:IFMID$(L$,5,3)="),Y"THEN3098
3067 GOTO3090
3085 A$=MID$(AM$,N*2-1,2):GOSUB2900:ER=1
3086 IFAANDCTHENER=0
3087 A=OPORO:GOSUB2800:RETURN
3090 P$="":ER$="OPERAND ERR":RETURN
3092 GOSUB3085:IFERTHEN3090
3093 P$=A$:RETURN
3094 GOSUB3085:IFERTHEN3090
3095 P$=A$+MID$(L$,S+2,2)+MID$(L$,S,2):RETURN
3098 GOSUB3085:IFERTHEN3090
3099 P$=A$+MID$(L$,S,2):RETURN
3100 REM ONE PASS TINY ASSEMBLER
3105 P$="":ER$="":N=0
3130 N=N+1:IFN>62THEN3145
3135 IFMID$(S$,1,3)<>MID$(MT$,N*3-2,3)THEN3130
3137 A$=MID$(OP$,N*2-1,2):GOSUB2900:OP=A:O=0
3140 ONVAL(MID$(JT$,N,1))GOSUB3155,3160,3162,3145,3145,3175,3145,3155,3155
3142 RETURN
3145 P$="":ER$="OPCODE ERROR"
3150 RETURN
```

```
3155 L$=MID$(S$,5,8):GOTO3000
3160 A=OP:GOSUB2800:P$=A$:GOSUB7800:P$=P$+A$:RETURN
3162 A=OP:GOSUB2800:P$=A$:RETURN
3175 IFOP=0THENGOSUB3200:P$=B$:RETURN
3176 IFOP=1THEN3145
3178 IFOP=2THENGOSUB3400:P$=B$:RETURN
3200 REM COMPRESS OPERAND
3210 B$="":FORJ=5TO12
3220 IFMID$(S$,J,1)<>" "THENB$=B$+MID$(S$,J,1)
3230 NEXTJ:RETURN
3400 REM CONVERT TXT OPERAND
3410 B$="":FORJ=5TO12
3420 IFMID$(S$,J,1)="/"THEN3460
3430 A=ASC(MID$(S$,J,1))
3440 GOSUB2800:B$=B$+A$
3450 NEXTJ
3460 RETURN
5600 REM MOVE DATA ROUTINE
5610 PRINT"&&MOVE DATA&&"
5615 INPUT"START ADDRESS";A$:GOSUB6695:LS=A
5620 INPUT"END ADDRESS";A$:GOSUB6695:LE=A
5625 INPUT"TO ADDRESS";A$:GOSUB6695:LT=A
5630 FORI=LSTOLESTEP255
5635 MC=255
5637 IF(LE-I)=0THEN5685
5640 IF(LE-I)<255THENMC=LE-I
5645 MF=I:MT=LT+(I-LS)
5650 POKE181,MC
5660 POKE182,MF-INT(MF/256)*256:POKE183,INT(MF/256)
5670 POKE184,MT-INT(MT/256)*256:POKE185,INT(MT/256)
5680 SYS(915)
5685 NEXTI
5690 GOTO6300
6200 REM INITIALIZE IMPORTANT VARIABLES
6204 X$="0123456789ABCDEF"
6208 LL$="222322332112"
6210 O$(0)="BRK;ORA0ERR;ERR;ERR;ORA1ASL1ERR;PHP;ORA2ASL ERR;ERR;ORA3ASL3ERR;"
6215 O$(1)="BPL<ORA4ERR;ERR;ERR;ORA5ASL5ERR;CLC;ORA6ERR;ERR;ERR;ORA7ASL7ERR;"
6220 O$(2)="JSR3AND0ERR;ERR;BIT1AND1ROL1ERR;PLP;AND2ROL ;ERR;BIT3AND3ROL3ERR;"
6225 O$(3)="BMI<AND4ERR;ERR;ERR;AND5ROL5ERR;SEC;AND6ERR;ERR;ERR;AND7ROL7ERR;"
6230 O$(4)="RTI;EOR0ERR;ERR;ERR;EOR1LSR1ERR;PHA;EOR2LSR ;ERR;JMP3EOR3LSR3ERR;"
6235 O$(5)="BVC<EOR4ERR;ERR;ERR;EOR5LSR5ERR;CLI;EOR6ERR;ERR;ERR;EOR7LSR7ERR;"
6240 O$(6)="RTS;ADC0ERR;ERR;ERR;ADC1ROR1ERR;PLA;ADC2ROR ;ERR;JMP8ADC3ROR3ERR;"
6245 O$(7)="BVS<ADC4ERR;ERR;ERR;ADC5ROR5ERR;SEI;ADC6ERR;ERR;ERR;ADC7ROR7ERR;"
6250 O$(8)="ERR;STA0ERR;ERR;STY1STA1STX1ERR;DEY;ERR;TXA;ERR;STY3STA3STX3ERR;"
6255 O$(9)="BCC<STA4ERR;ERR;STY5STA5STX9ERR;TYA;STA6TXS;ERR;ERR;STA7ERR;ERR;"
6260 O$(10)="LDY2LDA0LDX2ERR;LDY1LDA1LDX1ERR;TAY;LDA2TAX;ERR;LDY3LDA3LDX3ERR;"
6265 O$(11)="BCS<LDA4ERR;ERR;LDY5LDA5LDX9ERR;CLV;LDA6TSX;ERR;LDY7LDA7LDX6ERR;"
6270 O$(12)="CPY2CMP0ERR;ERR;CPY1CMP1DEC1ERR;INY;CMP2DEX;ERR;CPY3CMP3DEC3ERR;"
6275 O$(13)="BNE<CMP4ERR;ERR;ERR;CMP5DEC5ERR;CLD;CMP6ERR;ERR;ERR;CMP7DEC7ERR;"
```

```
6280 O$(14)="CPX2SBC0ERR; ERR; CPX1SBC1INC1ERR; INX; SBC2NOP; ERR; CPX3SBC3INC3ERR
6285 O$(15)="BEQ<SBC4ERR; ERR; ERR; SBC5INC5ERR; SED; SBC6ERR; ERR; ERR; SBC7INC7ERR
6300 REM PRINT MAIN OPTION SCREEN
6302 PRINT"&&MONITOR&&"
6303 PRINTTAB(5);"COPYRIGHT 1979 BY COMPU/THINK"
6305 PRINT:PRINT:PRINT:PRINT"FREE MEMORY STARTS AT ";"$";MO$:PRINT
6306 PRINT"M MOVE DATA IN MEMORY"
6307 PRINT"E EXECUTE PGM IN MEMORY"
6308 PRINT"D DUMP MEMORY"
6309 PRINT"U UNASSEMBLE PGM IN MEMORY"
6310 PRINT"C ADD TWO NUMBERS"
6311 PRINT"N CONVERT NUMBER"
6313 PRINT"A TINY PET-ASSEMBLER"
6314 PRINT:PRINT"ANY OTHER KEY RETURNS HERE"
6315 PRINT"HEXADECIMAL NUMBERS MUST BEGIN"
6316 PRINT"WITH A $ SYMBOL $A254 DECIMAL "
6317 PRINT"NUMBERS USE DIGITS ONLY 542    "
6318 PRINT"OCTAL NUMBERS BEGIN WITH AN O "
6320 GETA$:IFA$=""THEN6320
6325 IFA$="M"THEN5600
6330 IFA$="E"THEN6400
6340 IFA$="D"THEN7000
6350 IFA$="U"THEN6500
6380 IFA$="L"THEN7200
6381 IFA$="A"THEN1500
6382 IFA$="C"THEN8000
6383 IFA$="N"THEN8100
6399 GOTO6300
6400 REM EXECUTE ROUTINE IN MEMORY
6402 PRINT"&&EXECUTE ROUTINE&&"
6404 INPUT"ADDR";A$:GOSUB6695
6405 PRINT
6406 POKE901,INT(A/256)
6408 POKE900,A-INT(A/256)*256
6410 INPUT"AC ";A$:GOSUB6695
6412 POKE912,A
6414 INPUT"XR ";A$:GOSUB6695
6416 POKE913,A
6418 INPUT"YR ";A$:GOSUB6695
6420 POKE914,A
6425 SYS(890)
6430 PRINT:PRINT"RESULTS"
6432 PRINT
6434 A=PEEK(912)
6436 PRINT"AC ";A;TAB(15);"$";:B=A:GOSUB6655:PRINT
6438 A=PEEK(913)
6440 PRINT"XR ";A;TAB(15);"$";:B=A:GOSUB6655:PRINT
6442 A=PEEK(914)
6444 PRINT"YR ";A;TAB(15);"$";:B=A:GOSUB6655:PRINT
6499 GOTO6320
```

```
6500 REM UNASSEMBLE MEMORY ROUTINE
6501 PRINT"&&UNASSEMBLE ROUTINE&&"
6502 PRINT
6506 INPUT"ADDR";A$:GOSUB6695
6507 S=A
6508 PRINT"";:INPUT"INSTRUCTIONS";A$
6509 GOSUB6695:N1=A
6510 PRINT"HEXADECIMAL ASSEMBLER      DECIMAL":PRINT
6512 IFN1>20THENN1=20
6515 FORI=1TON1
6518 B=INT(S/256):GOSUB6655
6520 B=S-INT(S/256)*256:GOSUB6655
6522 PRINTTAB(24);RIGHT$("     "+STR$(S),5):PRINT"";TAB(4);
6525 PRINT" ";
6530 J=PEEK(S):K=INT(J/16):L=J-K*16+1
6535 M=ASC(MID$(O$(K),L*4,1))-ASC("0")+1
6540 N=VAL(MID$(LL$,M,1)):FORP=1TON
6545 B=PEEK(S+P-1):GOSUB6655
6550 NEXTP
6551 PRINTTAB(30);:FORP=1TON
6552 B=PEEK(S+P-1)
6553 PRINTRIGHT$("00"+RIGHT$(STR$(B),LEN(STR$(B))-1),3);
6554 NEXTP
6555 PRINT:PRINT"";
6558 PRINTTAB(12);MID$(O$(K),L*4-3,3);" ";
6560 B=PEEK(S+1)
6565 ONMGOSUB6675,6655,6680,6660,6685,6690,6650,6605,6610,6615,6620,6625,6630
6570 PRINT
6597 S=S+N
6598 NEXTI
6599 GOTO6320
6600 REM MISC.  ROUTINES FOR UNASSEMBLE
6605 GOSUB6660:PRINT",X";:RETURN
6610 PRINT"(";:GOSUB6660:PRINT")";:RETURN
6615 GOSUB6655:PRINT",Y";:RETURN
6620 PRINT"A";:RETURN
6625 RETURN
6630 IFB>127THENB=((NOTB)AND255)+1:B=-B
6635 T=S+2+B
6640 B=INT(T/256):GOSUB6655
6645 B=T-INT(T/256)*256:GOSUB6655:RETURN
6650 GOSUB6660:PRINT",Y";:RETURN
6655 PRINTMID$(X$,INT(B/16)+1,1);MID$(X$,B-INT(B/16)*16+1,1);:RETURN
6660 B=PEEK(S+2):GOSUB6655
6665 B=PEEK(S+1):GOSUB6655
6670 RETURN
6675 PRINT"(";:GOSUB6655:PRINT",X)";:RETURN
6680 PRINT"#";:GOSUB6655:RETURN
6685 PRINT"(";:GOSUB6655:PRINT"),Y";:RETURN
6690 GOSUB6655:PRINT",X";:RETURN
```

```
6695 IF"$"=LEFT$(A$,1)THEN6698
6696 IF"O"=LEFT$(A$,1)THEN900
6697 A=VAL(A$):GOTO6699
6698 K=1:GOSUB6900
6699 RETURN
6800 REM FIX BASIC BUG
6805 REM FIX BASIC BUG
6810 REM FIX BASIC BUG
6815 REM FIX BASIC BUG
6820 REM FIX BASIC BUG
6825 REM FIX BASIC BUG
6830 REM FIX BASIC BUG
6900 REM CONVERT HEX NUMBER TO DECIMAL
6905 A$=RIGHT$("0000"+A$,4):A=0
6910 FORJ1=KTO4
6915 C$=MID$(A$,J1,1):C=0
6920 IFC$>="A"ANDC$<="F"THENC=ASC(C$)-ASC("A")+10
6925 IFC$>="0"ANDC$<="9"THENC=ASC(C$)-ASC("0")
6930 A=A*16+C
6935 NEXTJ1
6999 RETURN
7000 REM DUMP MEMORY ROUTINE
7005 PRINT"&&MEMORY DUMP&&"
7010 PRINT:K=1
7015 INPUT"ADDR";A$:GOSUB6695:S=A
7020 PRINT"";:INPUT"# OF LINES";A$
7022 GOSUB6695:N=A:IFN>20THENN=20
7025 PRINT"\ADDR\ &&&&DECIMAL&&&& &&&HEXA&&& \CHAR":PRINT
7030 FORL=1TON
7032 PRINTTAB(7);
7035 FORI=1TO5:GOSUB7095
7040 PRINTRIGHT$("000"+RIGHT$(STR$(B),LEN(STR$(B))-1),3);:NEXTI
7045 PRINTTAB(23);:K=2
7050 FORI=1TO5:GOSUB7095
7055 GOSUB6655:NEXTI
7060 PRINTTAB(34);
7062 FORI=1TO5:GOSUB7095:GOSUB7400
7065 PRINTA$;
7070 NEXTI
7072 PRINT:PRINT" ";:T=S+(L-1)*5
7075 GOSUB6640:PRINT" !"
7090 NEXTL:GOTO7099
7095 B=PEEK(S+I+(L-1)*5-1):RETURN
7099 GOTO6320
7100 REM LOAD PROGRAM INTO MEMORY
7110 A$=LEFT$(P$,4)
7120 K=1:GOSUB6900:S=A
7130 FORI=5TOLEN(P$)STEP2
7140 A$=MID$(P$+"0",I,2)
7150 GOSUB6900
```

```
7160 POKES+INT(I/2)-2,A
7170 NEXTI
7180 POKE1,S-INT(S/256)*256
7190 POKE2,INT(S/256)
7199 RETURN
7300 REM DATA FOR STORED PROGRAM
7310 L9=0
7320 RETURN
7400 REM CONVERT HEX TO PET-ASCII
7410 IFB<32THENB=32
7420 IF(B>95)AND(B<160)THENB=32
7430 IFB>223THENB=32
7440 A$=CHR$(B)
7450 RETURN
7800 REM RESOLVE RELATIVE ADDRESS
7810 A$=MID$(S$,5,5):GOSUB6695
7820 A=A-(OA+2)
7830 IFA<0THENA=256+A
7840 GOSUB2800:RETURN
7900 REM POKE MEMORY
7910 FORI=1TOLEN(P$)STEP2
7920 A$=MID$(P$+"0",I,2)
7930 GOSUB6900
7940 POKES+INT(I/2),A
7950 NEXTI:RETURN
8000 REM CONVERT AND ADD HEX NUMBERS
8005 PRINT"&&ADD NUMBERS&&"
8010 PRINT:K=1
8015 INPUT"# ONE";A$:GOSUB6695:S=A
8017 PRINT:K=1
8020 INPUT"# TWO";A$:GOSUB6695:S2=A
8030 PRINT:K=1
8040 A=S+S2
8050 GOTO8120
8100 REM CONVERT NUMBER
8105 PRINT"&&CONVERT NUMBER&"
8110 PRINT
8115 INPUT"NUMBER ";A$:GOSUB6695
8120 PRINT:PRINT"DECIMAL = ";A
8140 PRINT
8145 GOSUB950:PRINT"OCTAL = ";B
8150 PRINT:PRINT"HEXADECIMAL = ";
8155 H=INT(A/256):L=A-(H*256)
8160 B=H:GOSUB6655
8170 B=L:GOSUB6655
8180 GOTO6320
```

```
********
DISKTEST
********


BASIC LISTING            PAGE 01


100 REM DISPLAY HEADER ON SCREEN
105 PRINT"&&DISKTEST&&"
110 PRINT"THIS TEST WILL WRITE DATA ON THE"
120 PRINT"DISKETTE DURING TESTING!         "
130 PRINT"FORMAT THE DISKETTE BEFORE USING"
140 INPUT"ENTER DISK #";D
150 PRINT"FORMAT TEST FILE"
160 Z$="":FORJ=1TO195:Z$=Z$+"X":NEXTJ
170 DEF FNA(I)=(TI-INT(TI/99)*99)/100
200 REM FORMAT LARGE RANDOM ACCESS FILE
210 DI$=CHR$(240)+CHR$(1)+CHR$(200)+CHR$(0)+"          "
220 $ODISK,D,"W","DISKTEST.DAT",DI$
230 FORI=1TO496
240 R$=Z$+RIGHT$("      "+STR$(I),5)
250 $WDISK,R$
260 NEXTI
270 $CDISK
280 PRINT"READING TEST FILE"
300 REM READ LARGE SEQUENTIAL FILE
310 $ODISK,D,"R","DISKTEST.DAT",DI$
320 FORI=1TO496
330 $RDISK,R$
340 X$=Z$+RIGHT$("      "+STR$(I),5)
350 IFX$<>R$THENPRINT"ERROR AT RECORD";I:STOP
360 NEXTI
370 $CDISK
380 PRINT"RANDOM READING"
400 REM READ LARGE RANDOM FILE
410 $ODISK,D,"D","DISKTEST.DAT",DI$
420 FORI=1TO100
430 N=INT(FNA(I)*496)+1
440 $RDISK;N,R$
450 X$=Z$+RIGHT$("      "+STR$(N),5)
460 IFX$<>R$THENPRINT"ERROR AT RECORD";N:STOP
470 NEXTI
480 $CDISK
490 PRINT"TEST SUCCESSFUL"
```

```
********
DISKCOPY
********


BASIC LISTING           PAGE 01


100 REM CLEAR AND SIZE THE MEMORY BY EXECUTING $HALT
101 REM BEFORE LOADING AND USING THIS PROGRAM
102 REM THIS PROGRAM IS SET UP FOR TWO DRIVES
103 REM USER MODIFICATIONS MAY BE NEEDED
104 X=PEEK(53)
105 POKE53,16
106 POKE51,16
107 POKE49,16
110 IFX>116GOTO120
112 IFX>96GOTO122
114 IFX>66GOTO124
116 IFX>56GOTO126
118 PRINT"DISKCOPY NEEDS MORE MEMORY(OR $HALT AND RELOAD)":END
120 TC=4:PE=116:GOTO135
122 TC=3:PE=96:GOTO135
124 TC=1:PE=56:GOTO135
126 TC=1:PE=56:GOTO135
135 DC=11*4096:REM COMMAND TABLE
138 DD=45051:REM ASC DEVICE #
140 DT=45055:REM ABSOLUTE TRACK #
142 DM=DC+06:REM TURN DISK MOTOR ON
145 DO=DC+39:REM TURN DISK MOTOR OFF
148 DP=49147:REM DEVICE SELECT REGISTER
150 D$="1":REM DISK DEVICE VARIABLE
155 T=0:REM ABSOLUTE TRACK VARIABLE
160 DB=16*256:REM START OF HOLDING AREA
165 DL=5120*(TC+1):REM LENGTH OF HOLDING AREA
170 DE=PE*256:REM END OF HOLDING AREA
175 BS=251:REM START BUFFER POINTER
180 BE=201:REM END BUFFER POINTER
200 REM PRINT OUT INSTRUCTIONS
210 PRINT"&&DISKCOPY&&"
220 PRINT"CLEAR THE MEMORY BY EXECUTING $HALT  "
225 PRINT"BEFORE LOADING AND USING THIS PROGRAM"
230 PRINT"THIS PROGRAM IS SET UP FOR TWO DRIVES"
240 INPUT"ENTER FROM DEVICE #";RD$:PRINT""
245 INPUT"ENTER COPY DEVICE #";CD$:PRINT""
250 PRINT"PRESS ANY KEY WHEN READY"
255 GETA$:IFA$=""THEN255
300 REM DISK COPY ROUTINE
310 FORTT=0TO39STEPTC+1
315 D$=RD$:GOSUB900:GOSUB600
320 FORT=TTTOTT+TC
325 PRINT"READING TRACK #";T;""
330 GOSUB800
335 NEXTT:GOSUB950
340 GOSUB500
345 D$=CD$:GOSUB900:GOSUB600
350 FORT=TTTOTT+TC
355 PRINT"COPYING TRACK #";T;""
```

```
********
DISKCOPY
********


BASIC LISTING           PAGE 02


360 GOSUB700
370 NEXTT:GOSUB950
375 NEXTTT
380 GOSUB500
390 GOTO200
500 REM TIMING LOOP
510 X=TI
520 IF(TI-X)<40THEN520
530 RETURN
600 REM SET BUFFER POINTERS
610 POKEBS,0:POKEBS+1,16
620 POKEBE,0:POKEBE+1,PE
630 RETURN
700 REM WRITE ABSOLUTE TRACK
710 POKEDT,T
720 SYS(DC+30)
730 RETURN
800 REM READ ABSOLUTE TRACK
810 POKEDT,T
820 SYS(DC+33)
830 RETURN
900 REM OPEN DISK
910 POKEDD,ASC(D$)
920 SYS(DM)
930 RETURN
950 REM CLOSE DISK
960 POKEDD,ASC(D$)
970 SYS(DO)
980 RETURN
```

```
***********
DENSITYCOPY
***********


BASIC LISTING          PAGE 01


10 REM DENSITYCOPY PROGRAM
15 REM COPYRIGHT 1979, COMPU/THINK
20 REM INITIALIZE VARIABLES          100
21 REM MAINLINE ROUTINE              200
22 REM PARSE SINGLE DENS DIRECTRY    300
23 REM DISPLAY SCREEN INFO           400
24 REM LOCATE NEXT TRACK             500
25 REM WRITE DUAL DENSITY TRACK      600
27 REM INITIALIZE EOR TRACK RTN      800
28 REM READ SINGLE DENS TRACK        900
99 GOTO200
100 REM INITIALIZE VARIABLES
102 GOSUB800
105 MR=49147:REM MOTOR ON REGISTER
110 MO=45089:REM READ TRACK INTO MEMRY
112 MB=45083:REM READ TRACK INTO BUFR
114 RS=46999:REM RESTORE HEADS COMND
115 TK=45055:REM ABS TRACK #
120 SB=00251:REM BUFFER START PTR
125 EB=00201:REM BUFFER END PTR
130 BF=36864:REM TRACK BUFFER
135 TP=45024:REM FILE NAME
140 DV=45051:REM DEVICE NUMBER
145 OP=45062:REM OPEN DISK RTN
150 CL=45065:REM CLOSE DISK RTN
155 AL=45074:REM ALLOCATE TRACK
160 WT=45080:REM WRITE DISK BUFFER
165 MF=45095:REM TURN MOTOR OFF
170 DIMNF$(39),RT(39),ID$(39)
175 BN$="                    "
180 BD$="        "
199 RETURN
200 REM MAINLINE ROUTINE
202 GOSUB100
205 GOSUB400
210 A=0:B=0:GOSUB900
215 GOSUB300
220 FORI=1TO39
225 IFRT(I)=1THEN235
230 NEXTI:GOTO299
235 NF$=NF$(I):RT=1:ID$=ID$(I):RT(I)=255:B=0:A=I:GOSUB900:DT=1
240 GOSUB500:IFA=255THENGOSUB600:GOTO220
245 B=1:GOSUB900:GOSUB600
250 DT=DT+1:GOSUB500:IFA=255THEN220
255 B=0:GOSUB900:GOTO240
299 END
300 REM PARSE SINGLE DENSITY DIRECTRY
301 PRINT"PARSING DIRECTORY                        "
305 SYS(634)
310 FORI=BF+25TOBF+975STEP25:N=N+1
```

```
***********
DENSITYCOPY
***********

BASIC LISTING          PAGE 02


320 FORJ=0TO15:NF$(N)=NF$(N)+CHR$(PEEK(I+J)):NEXTJ
330 RT(N)=PEEK(I+J)
340 FORJ=17TO24:ID$(N)=ID$(N)+CHR$(PEEK(I+J)):NEXTJ
350 NEXTI
360 RETURN
400 REM DISPLAY SCREEN INFORMATION
410 PRINT"&&DENSITYCOPY&&"
415 PRINT"PLACE SINGLE DENSITY DISK IN DRIVE #1"
420 PRINT"PLACE DUAL DENSITY DISK IN DRIVE #2"
425 PRINT"HIT ANY KEY TO COPY"
430 GETA$:IFA$=""THEN430
435 RETURN
500 REM LOCATE NEXT TRACK
501 PRINT"LOCATING NEXT TRACK                    "
505 RT=RT+1
510 FORIL=1TO39
515 IFNF$<>NF$(IL)THEN530
520 IFRT<>RT(IL)THEN530
525 A=IL:ID$=ID$(IL):RT(IL)=255:RETURN
530 NEXTIL
535 A=255
540 RETURN
600 REM WRITE DUAL DENSITY TRACK
601 SYS(MF)
603 PRINT"WRITING TRACK ON DRIVE #2             "
604 FORK=1TO50:NEXTK
605 SYS(634)
610 POKEDV,50:SYS(OP)
615 FORK=0TO15:POKETP+K,ASC(MID$(NF$,K+1,1)):NEXTK
620 POKETP+16,DT
625 FORK=1TO8:POKETP+16+K,ASC(MID$(ID$,K,1)):NEXTK
630 SYS(AL):SYS(WT):SYS(CL)
635 RETURN
800 REM INITIALIZE EOR DISK TRACK
810 FORI=0TO26
820 READP
830 POKE634+I,P
840 NEXTI
850 RETURN
860 DATA169,000,133,218,169,144
865 DATA133,219,160,000,162,020
870 DATA177,218,073,255,145,218
875 DATA200,208,247,230,219,202
880 DATA208,242,096
900 REM READ SINGLE DENSITY TRACK
901 REM A=TRACK, B=TRACK HALF
902 PRINT"                                       "
903 PRINT"READING TRACK ";A;" DRIVE #1           "
905 POKEDV,49
910 POKEMR,3:GOSUB999:POKEMR,3:GOSUB999:POKEMR,3:GOSUB999
```

```
***********
DENSITYCOPY
***********


BASIC LISTING           PAGE 03


918 SYS(RS)
920 POKETK,A
930 POKESB,0
940 POKESB+1,(9*16)+(B*10)
950 POKEEB,0
960 POKEEB+1,(9*16+10)+(B*10)
970 SYS(MO)
980 RETURN
999 FORK=1TO10:NEXTK:RETURN
```

```
********
DATAFILE
********


BASIC LISTING          PAGE 01


1 REM SIZE MEMORY AVAILABLE
2 TT=PEEK(53)
100 REM DATAFILE UTILITY PROGRAM
110 REM COPYRIGHT 1979, COMPU/THINK
120 REM DEFINE PROGRAM VARIABLES
130 DIMS$(TT)
135 CC$=""
140 X1$="&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&&"
145 FT$="FREE TRACK      "+CHR$(255)+"        "
150 ZQ$="*************************"
200 REM DISKMON MEMORY LOCATIONS
210 CS=44976:REM FILE FOUND SWITCH
220 CB=45008:REM BLOCK COUNT
230 CM=45010:REM MODE SWITCH
240 CD=43776:REM DISK DIRECTORY
250 CF=45024:REM FILE INFORMATION
260 CT=45055:REM ABSOLUTE TRACK
270 CC=45056:REM COMMAND TABLE
300 REM MAINLINE ROUTINE
310 GOSUB1400
320 PRINT"PRESS C TO COPY DATA FILE"
325 PRINT"PRESS D TO DISPLAY DATA FILE"
330 PRINT"PRESS P TO PRINT DATA FILE"
340 GETA$:IFA$=""THEN340
345 IFA$="C"THEN4000
350 IFA$="D"THEN4100
360 IFA$="P"THEN4200
370 GOTO300
1100 REM PRINT LINE OF LISTING
1110 $PRNT,PL$
1120 PL=PL+1
1130 IFPL<>60THENRETURN
1140 PL$="":$P,PL$:$P,PL$:$P,PL$
1145 $P,PL$:$P,PL$:$P,PL$:$P,PL$:$P,PL$
1150 GOTO1200
1200 REM PRINT PAGE HEADING
1205 PP=PP+1:PL=10
1220 PL$=LEFT$(ZQ$,LEN(DF$)):$PRNT,PL$
1230 $PRNT,DF$
1240 PL$=LEFT$(ZQ$,LEN(DF$)):$PRNT,PL$
1250 PL$=" ":$PRNT,PL$
1260 PL$=" ":$PRNT,PL$
1270 PL$="FILE INDEX = "+DI$+"             PAGE "+STR$(PP):$P,PL$
1280 PL$=" ":$PRNT,PL$
1290 PL$=" ":$PRNT,PL$
1299 RETURN
1400 REM FORMAT PET SCREEN
1405 PRINT"";X1$;""
1410 PRINT"&&DATAFILE&&"
1420 PRINTTAB(6);"COPYRIGHT 1979 COMPU/THINK"
```

```
1440 PRINTLEFT$(CC$,25);X1$;LEFT$(CC$,9):RETURN
1600 REM CLOSE FILE FOR MOD READ
1610 VL=PEEK(CB):VH=PEEK(CB+1):VT=PEEK(CF+16):$CDISK:RETURN
1699 RETURN
2300 REM READ FROM DATA FILE
2310 IF(SN<1)OR(SN>TJ)THEN2320
2315 S$=S$(SN):SN=SN+1:RETURN
2320 DX=D:DF$=SF$:GOSUB3600:SN=1
2330 $RDISK,R$:S$(SN)=LEFT$(R$+LEFT$(BB$,80),80):SN=SN+1
2335 IFLEFT$(R$,3)="EOF"THEN2350
2336 IFR$=""THEN2350
2340 IFSN<=TJTHEN2330
2350 GOSUB1600:SN=1:GOTO2315
3600 REM OPEN FILE FOR READ MOD
3610 $ODISK,DF,"R",DF$,DI$:DX$=DI$
3615 IFPEEK(CS)=255THENPRINT"FILE NOT FOUND ERROR":STOP
3620 IFVT=0THEN3699
3630 POKECF+16,VT:SYS(CC+15):SYS(CC+27)
3640 POKECB,VL:POKECB+1,VH
3699 RETURN
3800 REM MOD ONTO FILE FOR WRITE
3802 $ODISK,DT,"R",DT$,DI$
3803 IFPEEK(CS)<>255THEN3805
3804 $ODISK,DT,"W",DT$,DX$:RETURN
3805 $RDISK,R$:IFR$<>""THEN3805
3810 T=CD+(PEEK(CT)*25)-1
3815 FORI=1TO25:POKET+I,ASC(MID$(FT$,I,1)):NEXTI
3820 X=PEEK(CF+16):POKECF+16,X-1
3822 A=(PEEK(CB)+PEEK(CB+1)*256)-1:GOSUB3900
3825 POKECB,L:POKECB+1,H
3830 POKECM,2
3840 RETURN
3900 REM SEPARATE NUMBER INTO BYTES
3910 H=INT(A/256)
3920 L=A-(H*256)
3930 RETURN
4000 REM COPY ONE FILE TO ANOTHER
4001 VT=0:GOSUB1400
4005 PRINT"COPY OPTION"
4010 INPUT"FROM FILE NAME";DF$
4015 INPUT"FROM FILE DISK";DF
4020 INPUT"TO FILE NAME";DT$
4025 INPUT"TO FILE DISK";DT
4030 PRINT"COPYING FILE"
4035 GOSUB3600
4040 SN=0:FORI=1TOTT
4042 $RDISK,R$:IF(R$="")OR(R$="EOF")THEN4050
4043 S$(I)=R$:SN=I
4045 NEXTI
4050 GOSUB1600:GOSUB3800
```

```
********
DATAFILE
********


BASIC LISTING          PAGE 03


4055 FORI=1TOSN
4060 $WDISK,S$(I)
4065 NEXTI
4070 $CDISK
4080 IFSN=TTTHEN4035
4099 GOTO300
4100 REM DISPLAY DATA FILE ON SCREEN
4102 GOSUB1400:RL=39
4105 PRINT"DISPLAY OPTION"
4107 INPUT"ENTER FILE NAME";DF$
4110 INPUT"ENTER FILE DISK";DF
4115 $ODISK,DF,"R",DF$,DI$:TC=0:R$=""
4120 SN=0:GOSUB1400:PRINT"INDEX=";DI$;""
4125 GOSUB5000:IFPL$=""THEN4150
4130 SN=SN+1:PRINTPL$
4135 IFSN<12THEN4125
4140 PRINT"RECORD COUNT=";TC;" PRESS SPACE TO CONT"
4142 GETA$:IFA$=""THEN4142
4145 GOTO4120
4150 $CDISK
4155 PRINT"RECORD TOTAL=";TC
4160 GETA$:IFA$=""THEN4160
4199 GOTO300
4200 REM PRINT DATA FILE ON PRINTER
4202 GOSUB1400:RL=80
4205 PRINT"PRINT OPTION"
4207 INPUT"ENTER FILE NAME";DF$
4210 INPUT"ENTER FILE DISK";DF
4215 $ODISK,DF,"R",DF$,DI$:TC=0:R$=""
4220 PRINT"PRINTING":PP=0:GOSUB1200
4225 GOSUB5000:IFPL$=""THEN4250
4230 GOSUB1100:GOTO4225
4250 $CDISK
4252 PL$=" ":GOSUB1100:GOSUB1100
4255 PL$="RECORD TOTAL = "+STR$(TC):GOSUB1100
4299 GOTO300
5000 REM READ RECORDS FROM FILE
5010 IFR$<>""THEN5030
5020 $RDISK,R$:IF(R$="")OR(R$="EOF")THENPL$="":RETURN
5025 TC=TC+1
5030 IFLEN(R$)<RL+1THENPL$=R$:R$="":RETURN
5040 PL$=LEFT$(R$,RL)
5045 R$=RIGHT$(R$,LEN(R$)-RL)
5050 RETURN
```

```
*********
BLOCKTEST
*********


BASIC LISTING            PAGE 01


5 PRINT""
10 PRINT" MEMORY TEST PROGRAM:PSEUDO RANDOM DATA"
20 REM BY RICHARD TOBEY COPYRIGHT 6/16/1978
30 PRINT"":PRINT"   FOR FURTHER DESCRIPTION RUN 1000":PRINT""
40 GOSUB 700
100 INPUT"FROM BLOCK";A:INPUT"TO BLOCK";B
120 POKE 178,0:POKE 179,(A-1)*16:POKE 180,B*16-1
130 INPUT"# PASSES,0-255,0=CONTINUOUS";P
140 P=INT(P):IF P>255 GOTO 130
150 IF P>0 THEN C=0:GOTO 170
160 C=1:P=255:REM NO PASSES=CONT
170 INPUT"DELAY SECONDS";D
180 Q=0:R=P
190 POKE 181,Q
200 SYS(832):REM START FILL
210 IF D=0 GOTO 250
220 Y=D :REM DELAY
230 FOR X=1 TO 930:NEXT
240 Y=Y-1:IF Y<>0 GOTO 230
250 SYS(848):REM START TEST
260 IF PEEK(187)=0 GOTO 360 :REM ERROR TEST
265 PRINT""
270 PRINT"ERROR IN BLOCK";INT(PEEK(183)/16)+1
275 PRINT"( PAGE";PEEK(183);")","AT LINE";PEEK(182)
280 PRINT"":PRINT"TEST DATA    ";PEEK(185),"MEMORY ";PEEK(186):PRINT""
290 PRINT"NEXT LOCATION ?"
300 GET C$:IFC$="" GOTO 300
310 IF C$="Y" THEN SYS(908):GOTO 260
320 PRINT"":PRINT"RETEST?"
330 GET C$:IF C$="" GOTO 330
340 IF C$="Y" GOTO 180
350 END
360 Q=Q+1:PRINT"PASS";Q,"TIME ";TI$
370 R=R-1:IF R<>0 GOTO 190
380 IF C=1 GOTO 400:REM CONT FLAG SET
390 PRINT"TEST COMPLETE":GOTO 320
400 P=255:GOTO 180
700 REM SUBROUTINE TO LOAD MACHINE CODE
710 REM INTO THE SECOND CASSETTE BUFFER
720 FOR I=832 TO 912:READ W:POKE I,W:NEXT
730 RETURN
740 DATA 160,0,32,102,3,32,111,3
750 DATA 145,182,32,118,3,208,246,96
760 DATA 160,0,32,102,3,32,111,3
770 DATA 209,182,208,37,32,118,3,208
780 DATA 244,169,0,133,187,96,165,178
790 DATA 133,182,165,179,133,183,96,165
800 DATA 182,69,183,69,181,96,230,182
810 DATA 208,6,230,69,165,69,197,66
820 DATA 96,133,71,177,68,133,72,169
```

****
BLOCKTEST
****

```
830 DATA 255,133,73,96,160,0,76,92,3
1000 PRINT"":PRINT"THIS ROUTINE FILLS THE MEMORY WITH A
1005 PRINT"DIFFERENT PSEUDO RANDOM PATTERN
1010 PRINT"EACH PASS,THEN AFTER A DELAY PERIOD,
1020 PRINT"GENERATES THE PATTERN AGAIN AND
1030 PRINT"COMPARES IT TO THE CONTENTS OF MEMORY,
1040 PRINT"THUS CHECKING FOR POSSIBLE FAILURE
1050 PRINT"MODES,INCLUDING BAD CHIPS,STUCK
1060 PRINT"ADDRESSES,INADEQUATE REFRESH ETC.
1070 PRINT"":PRINT"  TO RUN THE PROGRAM
1080 PRINT"BLOCKS ARE 4K BYTES.  TO TEST THE 3RD
1090 PRINT"THRU 6TH BLOCKS OF MEMORY(16K ADDED)
1100 PRINT"USING SELECTS 2 THRU 5,(DEC 8192-24576)
1110 PRINT"FROM BLOCK IS 3,WHILE TO BLOCK IS 6.
1120 PRINT"THE # OF PASSES MAY BE FROM 1 TO 255
1130 PRINT"ENTERING 0 PASSES GIVES CONTINUOUS
1140 PRINT"TESTING
1150 PRINT"ALSO DELAY MAY BE ENTERED IN SECONDS
1160 PRINT"0=NO DELAY
```