6809 FLEX™ Diagnostics

MANUAL REVISION HISTORY

Revision	Date	Change
A	3/80	Original Release

DISCLAIMER

The supplied software is intended for use only as described in this manual. Use of undocumented features or parameters may cause unpredictable results for which Technical Systems Consultants, Inc. cannot assume responsibility. Although every effort has been made to make the supplied software and its documentation as accurate and functional as possible, Technical Systems Consultants, Inc. will not assume responsibility for any damages incurred or generated by such material. Technical Systems Consultants, Inc.. reserves the right to make changes in such material at any time without notice.

PREFACE

This software package contains memory diagnostics and disk diagnostic and repair utilities. They are intended to aid the user in detecting problems with computer memory and disk systems. Some of the utility programs assist in the recovering of data from damaged disks. It should be stressed that these programs do not perform miracles. There are certain types of failures which are very difficult for any diagnostic to detect, or for any repair program to correct. It was intended that the programs in this package detect many common problems rather than only a few rare problems.

This manual is divided into two main sections. The first deals with the memory diagnostics, and the second deals with the disk diagnostic and repair utilities.

In the section on memory-diagnostics, there is a discussion of memory diagnostics and computer memory in general. It is recommended that this subsection be read even if you are already familiar with memory diagnostics and computer memory since several terms are defined here. The next subsection gives an overview of the memory diagnostics in this package. A troubleshooting guide is next, which outlines some procedures to follow when hunting for a memory problem. A detailed description of each diagnostic completes the section on memory diagnostics.

In the disk utility section, we have a general discussion of disks and their problems, followed by an overview of the utilities in this package. A troubleshooting guide for disk problems then discusses the symptoms of disk problems and gives some general hints on tracking down a problem. This is followed by detailed descriptions of the disk utilities. Lastly, a "Case Studies" section demonstrates how the disk utilities may be used in detecting, and sometimes correcting, problems.

TABLE OF CONTENTS

INTRODUCTION TO MEMORY DIAGNOSTICS

INTRODUCTION TO MEMORY DIAGNOSTICS

Memory diagnostics are used to determine if a problem with the computer's memory exists. If a problem is detected, the diagnostic should also give information to assist the user in determining which parts must be replaced or repaired. We shall not go into great detail in discussing how a computer's memory operates and why it fails as this would involve discussing electronics and semiconductor physics. Only enough background information will be given so that the user can make a reasonable decision as to which diagnostic to run, and how to interpret the messages given by the diagnostic. The aim of this introduction is to give the reader a general feeling for memory problems and memory diagnostics.

Memory Diagnostics In General

A "perfect" diagnostic does not exist. That is, there is no diagnostic that can detect all possible memory problems and accurately report on them. Such a diagnostic would have to be designed based on detailed knowledge of the internal physical and electronic structure of the components used in the construction of the memory that is going to be tested. With the large number of memory chips and circuit boards on the market, the writing of a diagnostic that will work correctly on each of them is impossible. Instead, more general diagnostics are written which are applicable to a large variety of memory types and organizations.

Over the years, certain diagnostic techniques have shown themselves to be effective in detecting and isolating failures even when used on different types of memory. Some of these techniques that have been incorporated into the memory diagnostics in this package. However, since no diagnostic is "perfect", there will always be certain types of failures that a given diagnostic will not detect correctly. Therefore, it is good practice to run several different diagnostics just in case the first ones were not able to detect an error. It is also good practice to let a diagnostic run for several minutes so that it has a chance to detect "intermittent" errors; that is, those errors that do not fail every time, but only once in a while.

Memory diagnostics work by storing known data into memory and then later checking to see if the data is the same as was stored there. If the pattern is the same, then the diagnostic assumes that the memory is working correctly. The diagnostic may then try different data, or it may use the same data over again. If the data does not match what was stored there, the diagnostic assumes that the memory has failed to store the information correctly. It knows what the data is supposed to be, and it knows what it actually is. It also knows into which memory location the data was stored. All of this information is reported back to the user's terminal. With this information, and some other information provided by the manufacturer of the memory, the user can usually determine which circuit board or memory component is not working correctly.

Most memory diagnostics differ only in the data that is used to test the memory. The different types of memory diagnostics will be discussed a little bit later in this introduction.

Memory in General

Computer memory can be contructed from various materials and electrical components. There is magnetic "core" memory, semiconductor memory, magnetic bubble memory, and other less widely known forms of computer memory. The type of memory in the computer determines which diagnostics are most effective in detecting problems. For example, diagnostics which detect problems specific to core memory are not that useful when testing magnetic bubble memory. The most common form of memory found in microcomputers is semiconductor memory, often called memory "chips" because it is packaged in the form of integrated circuits. Other types of memory are found so rarely in microcomputers that no further discussion of them is warranted.

There are two forms of semiconductor memory: static memory and dynamic memory. In static memory, once a value is stored, it will stay there until changed as long as the machine is turned on. In dynamic memory, the data will fade away unless it is "refreshed" periodically. Normally, the refresh is performed automatically by additional circuitry on the memory board. The advantages of dynamic memory over static memory are that it uses less power and it can be made denser, that is, more data can be stored on a memory chip. The user should be aware of which type of semiconductor memory is in the machine. If it is static, there is no need to run diagnostics which detect problems specific to dynamic memory, like the test in this package called DYNAMIC.

Even with dynamic memory, only a relatively small amount of information can be stored on an individual memory integrated circuit. In addition, the internal structure of most memory chips is such that only one bit at a time is accessible. To get around these problems, computer memory is usually made up of several memory chips. The problem of being able to read only one bit at a time is solved by having eight chips act together, each of them storing one bit of the eight-bit byte. For lack of a better term, we will call this "parallel organized memory". Thus, one segment of memory consists of eight memory chips acting in parallel to hold the eight-bit bytes. Larger memory capacity is realized by adding more memory segments, that is, more sets of eight chips. Note that in parallel organized memory, each chip holds one bit from each byte. One chip holds the leftmost bit from each byte, another holds all of the bits that are second from the left, and so forth. Some of the memory diagnostics are designed to take advantage of this organization. By changing all of the bits in a byte, the diagnostic is really changing only one bit in each memory chip. Thus, it can test eight chips simultaneously.

Most microcomputer memory is parallel organized memory. This is because most memory chips made today are those that can handle only one bit at a time. Some machines contain small scratchpad memories for use by the monitor. These memories are usually on a single chip and can

process all eight bits at once. The diagnostics included in this package will run on any organization of memory, but some may not be very efficient on some organizations.

Types of Memory Failures

When memory fails, one of two cases exist: a bit is a zero when it should be a one, or it is a one when it should be a zero. The first case is called a "bit drop"; the second, a "bit pick". In addition, memory failures are said to be "hard" failures when they can be reproduced at will; that is, when a given pattern will always demonstrate the failure. "Soft" errors, however, cannot always be reproduced.

Soft errors may be temperature dependent. This means that the memory will fail only when it is "hot" or only when it is "cold". Memory that fails when it is cold will fail only for a short time after the machine is turned on. Once the machine warms up, the memory will run without error. Memory that fails when it is hot will not fail until after the machine has had time to warn up. When a problem is suspected, the memory should be checked out under both conditions, hot and cold.

A memory failure is said to be "pattern sensitive" when only specific test patterns expose the error. For example, a bit may pick only if all of the bits surrounding it in the chip are ones. Many diagnostics try more than one pattern in an attempt to detect such sensitivity.

The "intermittent" error is one that appears seemingly at random. Some times the memory will fail with a given pattern, and other times it will not fail with the same pattern. Such problems are extremely difficult to detect, even with the best diagnostics.

There are other sources of memory errors than the memory chips themselves. A memory board contains circuits that determine which chips are to be read or written; this is called the "select logic". Also, some boards contain their own power regulators. Dynamic memory boards may contain additional logic to perform the refresh operation. Failure in these circuits may result in very peculiar errors. For example, refresh failure may result in all of the memory on the board being cleared. The troubleshooting hints later on in this manual will mention more about failures in these circuits.

Types of Memory Diagnostics in General

The ideal memory diagnostic is one that would try all possible data patterns when testing a memory chip. For a memory chip containing 2048 bits, this would mean trying an extremely large number (3 followed by 616 zeroes) of patterns. Clearly, this is not practical. There are, however, some patterns which are used frequently because they detect the most common errors. These include: all zeroes, all ones, walking zero, and walking one. In addition, a "random" pattern is often used in the

hope that it will eventually find a failing pattern.

The "all zeroes" pattern is quite simple. The entire memory chip is cleared, that is, set to zero. This is used to detect hard bit picks, that is, any bit that cannot be set to zero. Conversely, the "all ones" pattern sets every bit in the chip to a one. This is to detect hard bit drops, that is, any bit that cannot be set to a one.

The "walking zero" pattern is more of a technique than an actual pattern. The pattern itself has every bit in the chip, except one, set to a one. Thus there is one bit that is different from the all of the other bits in the chip. This is an attempt to detect weak bits that may be unduly influenced by their neighboring bits in the chip. Once the pattern is written and checked, the data pattern is shifted so that some other bit is now the only one that is different. This continues until each bit in the chip has, at some time, been different from all of the others. Thus, the "different" bit is said to have "walked" through the memory chip. The "walking one" test is the complement of the "walking zero" test in that all of the bits in each chip, except one, are zero. The process of "walking" the bit through the memory chip is the same.

A random pattern is one that is generated by a pseudo-random number generator. A series of pseudo-random numbers is generated and stored in memory as the test pattern. After the pattern has been checked for errors a new sequence is generated and used as the test pattern. This process is repeated over and over in the hope that pattern sensitive errors will be uncovered.

There is one type of test that doesn't actually check the ability of a chip to retain data, but rather checks the select logic and data paths. This is the "convergence" test. The basic goal of a convergence test is to determine if more than one bit is changing in response to the writing of data. Causes of such failures are discussed in the troubleshooting guide.

THE MEMORY DIAGNOSTICS IN THIS PACKAGE

This diagnostic package contains six memory diagnostics. There is a zeroes and ones test (QUICK), a random pattern test (RANDOM), a convergence test (CONVERGE), a dynamic memory dropout test (DYNAMIC , a walking zero test (WALKO), and a walking one test (WALK1). The two most useful tests are QUICK and RANDOM. CONVERGE and DYNAMIC will probably not be used frequently unless the user suspects that a convergence problem or bit dropout problem actually exists. WALKO and WALK1 do not take advantage of the parallel organization of most microcomputer memory; thus they run slowly. Their most frequent application would be in testing small scratchpad memory chips, such as the Motorola MC6810, which are not parallel organized memory devices.

All six diagnostics are written to be position independent; that is, they will run correctly regardless of where they are loaded in user memory. Normally, they are loaded into the FLEX™ Utility Command Space. However, they may be moved to other areas of memory so that the memory in which FLEX normally resides may be tested. This is discussed later in "Relocating Memory Diagnostics".

To aid in analysing failures, each diagnostic, when an error is detected, prints the failing pattern in binary. Thus, any bit drop or bit pick is immediately apparent, and the number of the failing bit can be readily determined.

System Dependencies

The memory diagnostics are designed to run under the FLEX operating system. They use FLEX subroutines to decode their arguments. Once running, however, the diagnostics have no need for FLEX. This allows the memory in which FLEX normally resides to be tested. While running, all terminal input and output is performed through the system monitor ROM. As released, the diagnostics are configured to interface to the SWTPc® monitor SBUG-E®. Adapting to other monitors is discussed in the section "Adapting Memory Diagnostics to Other Monitors".

Some 6809 systems use a memory mapping device to make all of the memory in the machine appear to be contiguous, even if it is not. Hardware is installed which translates the address that the CPU references (called the "logical address") to the actual address of the memory board (called the "physical address"). The diagnostics cannot detect this mapping and thus will always report logical addresses instead of physical addresses when issuing messages.

As each diagnostic runs, it issues a character to the terminal as an indication of having completed one pass of the test. The number of such characters that the diagnostic will print per line is determined by the FLEX TTYSET width value. Since the diagnostics do not use FLEX for

---

FLEX is a trademark of Technical Systems Consultants, Inc.
SWTPc and SBUG-E are trademarks of Southwest Technical Products Corporation.

output, they memorize the width value and act on it themselves. If the width value is zero, it is assumed that the user's terminal will automatically go to a new line whenever the end of the current line is reached. If the user's terminal does not have this capability, set a width value using the TTYSET command to prevent the characters printed by the diagnostic from, running off of the right side of the terminal.

Relocating Memory Diagnostics

The only time that a diagnostic would have to be relocated would be when it is desired to test the area of memory in which the operating system resides. To facilitate this relocation, a utility called RUN has been added to the diskette. To test the operating system area with a diagnostic, type:

RUN,O,diagnostic name,COOO,DFBF

"Diagnostic name" is the name of the diagnostic being run. Any of the memory diagnostics, except DYNAMIC, may be specified. DYNAMIC cannot be used to test all of the operating system area because it would destroy scratch cells used by the SBUG-E monitor on SWTPc systems. The arguments COOO and DFBF specify the FLEX area of memory. Do not test the area DFBF through DFFF on systems with an SBUG-E monitor as this destroys data used by the monitor in performing input and output.

When testing the operating system area, the message FLEX ASSUMED OVERWRITTEN will be printed. This is purely an informative message and does not affect the running of the diagnostic. When the diagnostic is stopped, control will be returned to the monitor ROM since FLEX was destroyed during the test. It is necessary to reboot the system after testing the operating system area.

The run-time stack for each diagnostic is located immediately after the diagnostic itself. All diagnostics, except CONVERGE, need 32 bytes for stack space; CONVERGE needs 256 bytes. There is no need to be concerned about the stack when relocating a diagnostic to test the operating system area.

Adapting Memory Diagnostics to Other Monitors

As mentioned earlier, the diagnostics in this package are designed to run on a system using the SWTPc monitor SBUG-E. To aid in conversion to systems which run FLEX, but with a different monitor, the calls to the monitor routines are vectored. The vectors are the same in all of the memory diagnostics. Currently, each vector is an indirect jump. These may be replaced by indirect jumps or extended jumps, as required.

There are four vectors. Following is a description of each of them. The address given is that of the jump instruction itself, not the bytes to be patched. Those bytes to be patched will differ depending on whether extended or indirect jumps are used to interface with the monitor routines.

Input Character ($C103)

This routine reads one character from the terminal, returning it in the A-accumulator. The parity bit must be removed by the routine. The B, X, Y, and U registers must be preserved.

Output Character ($C107)

This routine outputs the character in the A-accumulator to the terminal. The B, X, Y, and U registers must be preserved.

Check for Character ($C10B)

This routine checks for a character having been typed on the terminal. The "Z" status bit is used to communicate the result of the check. If "Z" is on (BEQ instruction will branch), a character was not typed. If "Z" is off (BNE instruction will branch), a character was typed. The character should not be read by this routine. The B, X, Y, and U registers must be preserved. If the monitor does not have such a routine, make the following patch:

Starting at $C10B: 1A 04 39

With this patch, the test will still run, but it can be stopped only by resetting the machine; typing a character will not stop the diagnostic.

Return to Monitor ($C10F)

This returns control to the monitor ROM. It is used only when the diagnostic is stopped after having tested the operating system area.

MEMORY TROUBLESHOOTING GUIDE

Introduction

This portion of the manual gives hints on using the memory diagnostics in this package. Some of the suggestions are directed to those users who repair their equipment themselves. If you are not one of these persons, it suffices to merely determine that a problem exists, and then have the machine repaired by the supplier or manufacturer.

Symptoms of a Problem

There are some events that may occur when using the computer which indicate that memory might be failing. While some of these may also be caused by errors in a program, the possibility of memory failure should be kept in mind. Here are some things which might be an indication of failing memory:

1) Working programs suddenly do not work properly.
   If a program that is known to run correctly suddenly starts producing unpredictable results, it may be because the program has been destroyed in memory by a memory failure. Of course, it is possible that a previously unknown error in the program is causing the failure. If the program that has failed is one that you have recently modified, it might be that you introduced an error when making the modification. Also, the program may be reacting to unrecognized input; the "garbage in, garbage out" phenomenon. However, if the program has been running solidly for a long time, and suddenly starts to fail for most input, then memory failure should be considered as a possible cause. It does not take long to run some diagnostics to check, the memory, and it could avoid trouble later on.

2) Running the same program more than once, with the same data, produces different results.
   This is a more reliable indicator of hardware problems. Of course, an uninitialized variable in the program may also give this symptom. However, if a program which has run reliably in the past starts producing inconsistent results, memory failure may be indicated.

3) Data files being altered.
   If the data in a file does not correspond to what was written to it, it is possible that the data was altered in the buffer by a memory failure. One should be sure, however, that a program error did not write the data incorrectly. Some additional clues may be discovered by analysing the changes in the file. If characters in the file change their "case", (upper case letters become lower case, or the converse), this would indicate a change in one bit of the character, which might be caused by memory failure. If a data character is changed to another, and the difference is only in one bit, then memory problems should be suspected. For example, a "G" (binary 01000111) may be turned into an "F" (binary 01000110).

The above list is not exhaustive. However, any time that "strange things" happen in a program, one must not immediately blame the program. While it would be foolish to immediately spend an hour running memory tests every time that something unexpected happens, memory failure should be considered a possibility, especially if "solidly running" programs start to fail.

Tracking Down the Problem

If you suspect a particular type of memory problem, then it makes sense to choose a diagnostic that is tailored to finding that type of problem. On the other hand, if you are not sure that there is a problem with memory, but you suspect that something is wrong, no one diagnostic may be enough to isolate the problem. Several diagnostics may have to be run before the problem is found, or before it becomes evident that memory is not at fault. The following procedure is a good starting point in tracking down problems.

1) Run the diagnostic QUICK, without parameters, for 20 passes (20 plus signs printed). If no failure is uncovered, then:

2) Run the diagnostic RANDOM, without parameters, for 20 passes (20 plus signs printed). If the failure does not yet show up, then:

3) Run QUICK and RANDOM in the operating system area for 20 passes each. (See the section "Relocating Memory Diagnostics" for the proper procedure.) If an error has not yet been uncovered, then memory is in reasonably good shape. If there is a memory problem, it is very intermittent, very pattern sensitive, or a convergence problem. It may also be a refresh problem with dynamic memory. The following steps attempt to detect these problems.

4) If you do not have dynamic memory in your computer, skip this step; if you do, then run the diagnostic DYNAMIC for 15 minutes. Do not try to run this test in the operating system area. If no error is detected, then the refresh logic is probably not failing.

5) Run the diagnostic CONVERGE for 5 passes (10 characters are printed on the screen). If no error is detected, then run 5 passes of CONVERGE in the operating system area. If no error is detected at this point, then the address and data paths are probably working correctly.

6) Run WALK0 and WALK1 for 10 passes each (10 plus signs printed). If no error is detected, run 10 passes of each in the operating system area.

If, after all of this, no error has been uncovered, then either there is no problem, or the problem is temperature sensitive or very intermittent. If all of the above tests were run after the computer had been warmed up, then they should be run again while the computer is cold. To do this, turn off the computer and let it cool down for about 15 minutes. Turn on the machine, boot up the system, and immediately

perform one of the steps above. If no error is detected, turn off the machine, let it cool down, then turn it on and perform the next step. If this does not find the problem, then it is probably not a temperature dependent problem.

If the diagnostic does not run correctly (gives meaningless messages, returns to the monitor, does not Print pass indicators, etc.), it is possible that it is running in the area of memory that is bad. Normally, the diagnostics run in the FLEX Utility Command Space. If they do not run, try to relocate them and test the operating system area. Note, however, that the RUN utility, used to relocate diagnostics, also runs in the Utility Command Space, so it too might not run correctly. If it also fails, this is an important clue as to where the bad memory is located.

For extremely intermittent or pattern sensitive problems, a last resort measure is to run RANDOM for many hours. For dynamic memory, DYNAMIC should also be run for many hours. If none of these techniques uncover the problem, and the problem really is a memory problem, then it would require specialized troubleshooting methods far beyond the scope of these diagnostics.

If You Discover A Failure...

When one of the diagnostics detects a memory failure, it prints an error message. What to do next depends on your knowledge of computer hardware. If you do not perform your own maintenance and repair, it would be best to contact the dealer or manufacturer of your computer for advice. They will probably tell you where you can have the machine serviced. When you take the machine for service, take along a copy of the error message to show them.

If you intend to attempt repair yourself, you should be aware of the possible causes of the failure. A bad memory chip is usually the first suspect, but there is more to computer memory than the memory chips themselves. Here is a list of possible sources of memory failure:

1) Bad memory chip
2) Bad contact between memory board and mother board
3) Bad seating of memory chip in socket
4) Corrosion on pins of memory board or memory chip
5) Corrosion in the socket
6) Cold solder joint
7) Solder bridge (manufacturing defect)
8) Broken trace on circuit board
9) Chip select (address decode) circuitry on the board
10) Data or address buffer chips on the board
11) Refresh logic for dynamic memory
12) Regulator or other components (resistors and capacitors)

If CONVERGE or DYNAMIC was the test that discovered the problem, this could be a clue as to the cause. Recall that these two tests check for very specific problems, usually caused by failures in specific

circuitry. We will deal with these later.

Assuming that CONVERGE or DYNAMIC was not the diagnostic that uncovered the error, the next step is to isolate the failure to a single circuit board or memory chip. The error message should give the address that failed and the bit pattern that failed. From the address, determine which memory board contains the failing memory chip. Keep in mind that some machines re-map memory so that the address printed may not correspond to that for which the memory board is wired. The manufacturer should have given you information on how the mapping is accomplished, enabling you to determine which memory board contains the failing chip. From the "expected" and "received" values printed in the message, and from manufacturer's documentation on the organization of the memory board, you should be able to determine which chip is suspect.

Before replacing the suspected chip, there are two things you should try.

1) With the power off, move the memory board to another slot, if one is available. The problem may be caused by bad seating of the memory board on the mother board. If the test still fails, then:

2) If the memory chips are in sockets, switch the one that is failing with another one. Turn on the machine and rerun the diagnostics. One of three things should happen.

   a) If a failure occurs, it should be different because you have moved the suspected chip. If the failure indicates that the bad chip has failed, it is the cause of the problem; replace it.

   b) If the diagnostics no longer fail, the problem might have been the seating of the chip in the socket.

   c) If the problem. stays the same, then the chip probably was not at fault. A possibility may be dirt or corrosion in the socket, or a cold solder joint.

If you have convinced yourself that the problem was due to a faulty memory chip, and you have replaced it, but the problem did not go away, then try replacing it with another chip. It might be the case that the spare chip used to replace the bad one was also bad. If it still fails, then the problem is probably not due to a chip.

If DYNAMIC was the test that failed, the problem could be in the refresh logic. This would definitely be indicated if more than one bit has failed in the byte. If only one bit has failed, it might be that the memory chip is weak and has difficulty holding the data between refresh cycles. If only one bit has failed, then the first step is to assume that the chip might be bad, and to follow the procedure outlined above. If it becomes apparent that the chip is good, then you will have to assume that the refresh logic is at fault.

If CONVERGE is the diagnostic that has found the error, then the problem could be located in the chip select logic on the memory board. Memory chips have been known to short out internally in such a way that only a convergence test can detect the error, so the chip could still be at fault. The troubleshooting techniques outlined above for bad chips can be applied to convergence problems to determine if, indeed, such a failure within a chip has taken place. If the chip itself is not bad, then the selection logic, or perhaps the address or data buffer chips are at fault. It is also possible that a solder bridge exists, but this would be a manufacturing defect, not something that would suddenly appear on a working board. If you have built the board from a kit, however, solder bridges are something for which you should check.

General Hints

Tracking down memory problems is not often a simple procedure. In many cases, it is like solving a puzzle. Intermittent failures are probably the most baffling errors to uncover. Here are some hints which may help to uncover an intermittent error.

1) Make use of as much information as possible from the original failure. If you first suspected a memory problem because the data in a file was changed, then examine the changed data and try to determine which bit failed.

2) Make use of idle machine time to run diagnostics. If you are taking a break from your work for a few minutes, let QUICK or RANDOM run while you are away from the machine. You might uncover an error and avoid trouble later on when the problem becomes worse.

3) When you turn on the machine, run a few passes of QUICK or RANDOM; just to make sure that the shock of turning on the machine did not cause a weak or intermittent memory chip to fail completely.

4) Keep a notebook near the machine which you can use to record your progress as you troubleshoot a problem. Write down error messages and the steps you took to try to isolate the problem. If the problem is intermittent, it might return at some time in the future, and the notebook will tell you what steps you have already taken in trying to find the problem. Extremely intermittent errors occur so far apart that you should not trust your own memory in trying to recall what you did in the past.

5) If you do have a very intermittent problem, such that you have only one message from a diagnostic, try switching that memory chip with another one on the board (if they are in sockets). The next time that you get an error, which may be many days away, check to see if the new error points to the chip that was moved. If so, it was probably bad and you should replace it.

MEMORY DIAGNOSTIC DESCRIPTIONS

Program Name: CONVERGE
Program Type: MEMORY DIAGNOSTIC

PURPOSE:

CONVERGE is a form of convergence test, used primarily to detect address or data lines which are shorted together.

Calling Sequence:

CONVERGE,starting-address,ending-address

where:

"starting-address" is the lowest address in the block to be tested.

"ending-address" is the highest address in the block to be tested. If none is specified, the FLEX "Memory End" value is used.

If no arguments are specified, the block from "0000" through the FLEX "Memory End" value is tested.

METHOD:

Each pass of CONVERGE is divided into two major sections, an intra-byte check and an inter-byte check.

The intra-byte check determines if other bits in a byte change state when they are not specifically modified. The intra-byte check operates by performing a "walking one" test on each byte. The right-most bit of each byte is set to a "1" while the rest of the byte is cleared. The bytes are then checked and any errors reported. The next bit in each byte is then tested in the same way. After all eight bits have been tested, a plus sign is printed and the second section of the test, the inter-byte test, is performed.

The inter-byte test determines if data changes in some byte other than the one being addressed. CONVERGE uses a binary searching algorithm to reduce the time for the inter-byte test. At the start of the test, the block of memory to be tested is divided in half. The first half is cleared and the second half is set to an "all ones" pattern (hex FF). The first half is then checked for any bits having been set due to the writing of the "FF" pattern in the second half. If none are found, it is assumed that changing the second half of the block has no effect on the first half. The roles of the two halves are then reversed, the second half being cleared and the first half being set to "FF". The second half is then checked. If no errors are detected, it is assumed that the two halves do not affect each other. Each half is then separately tested, i.e. each half is itself divided into two parts and

the test performed on these halves. The test continues, recursively testing smaller segments of memory which have already been shown to be independent of the rest of the memory block. When the entire block has been tested, a minus sign is printed and the test starts over with the intra-byte test.

When an error is detected in the second part, the inter-byte test, CONVERGE attempts to isolate the error. The byte that was in error is cleared. Then that half of memory that was set to "FF" is again set to "FF" except that the byte that failed is checked after each byte is stored. If the error recurs, the last address which was set to "FF" is assumed to be the one that caused the error. Both the failing address and the last address which was changed are reported. If the error does not recur, the error is reported as an "intermittent" error, and only the address that changed is known and reported.

The test repeats until stopped by the user. The keyboard is monitored by the diagnostic, and whenever any character is typed, the test terminates, returning to the operating system.

MESSAGES:

ERROR IN ADDRESS
: An invalid hexadecimal character was detected in either "starting-address" or "ending-address". The test is aborted.

"CONVERGE" IS IN THE TEST AREA
: The diagnostic "CONVERGE" resides in the block of memory being tested. The test is aborted.

STACK IS IN TEST AREA
: The run-time stack used by CONVERGE is in the block of memory being tested. The test is aborted.

LAST < FIRST
: "Ending-address" is less than "starting-address". The test is aborted.

FLEX ASSUMED OVERWRITTEN
: The block of memory being tested contains all or part of the FLEX operating system. If the diagnostic is stopped from the keyboard, control will go to the monitor instead of FLEX.

ADDRESS xxxx, EXPECTED: nnnnnnnn, RECEIVED: nnnnnnnn
: An error was detected while testing the byte at address "xxxx". The data pattern stored is that indicated by the EXPECTED value. The incorrect data pattern read back is that indicated by the RECEIVED value. Both values (nnnnnnnn) are given in binary. This error message is produced by the intra-byte section of the diagnostic.

STORE ADDRESS: xxxx, READ ADDRESS: xxxx, DATA: nnnnnnnn
When "FF" was stored in the address given by "store address", the value at "read address" changed from zero to that given by "data". The data value is reported in binary. This error message is reported by the inter-byte section of the diagnostic.

INTERMITTENT ERROR, ADDRESS: xxxx DATA: nnnnnnnn
An error was detected during the inter-byte section of the test, but the error could not he isolated. The data at "address" changed from zero to that specified by "data". The data value is given in binary.

TESTING COMPLETED
The diagnostic has terminated, returning to the system.

REMARKS:

CONVERGE takes 5 seconds to test 4K (4096 bytes) of memory with a 1 MHz system clock. One-third of this time is due to the intra-byte test, and two-thirds is due to the inter-byte test. For 32K, the test takes 40 seconds per pass.

Program Name: DYNAMIC
Program Type: MEMORY DIAGNOSTIC

PURPOSE:

DYNAMIC tests a block of dynamic RAM for bit dropout under conditions such that the memory is not accessed for a period of time. Under these conditions, only the hardware refresh logic keeps the data current.

Calling Sequence:

DYNAMIC,starting-address,ending-address

where:

"starting-address" is the lowest address in the block to be tested.

"memory size" is a decimal number indicating the size of the block being tested in terms of "K" (1024 bytes); e.g. 16 implies 16K of memory (16*1024 bytes). If none is specified, 32 is assumed (311-768 bytes).

If no arguments are specified, 32K starting at "0000" is tested.

METHOD:

When invoked, DYNAMIC writes an "all ones" pattern (hex FF) in the block being tested. The data is checked and any errors reported. The test then delays approximately 10 seconds and checks the data again. The data pattern is never rewritten. If an error is detected, it is reported, and the failing byte is again set to hex FF. The test runs until stopped by the user. While delaying, DYNAMIC monitors the keyboard. If any character is entered, the test terminates immediately, returning to the operating system.

MESSAGES:

ERROR IN ADDRESS
The "starting-address" was not a legal hexadecimal number. The test is aborted.

ERROR IN SIZE
A non-decimal digit was detected in "memory size". The test is aborted.

"DYNAMIC" IS IN THE TEST AREA
The diagnostic "DYNAMIC" resides in the block of memory being tested. The test is aborted.

STACK IS IN TEST AREA
: The run-time stack used by DYNAMIC is in the block of memory being tested. The test is aborted.

FLEX ASSUMED OVERWRITTEN
: The block of memory being tested contains all or part of the FLEX operating system. If the diagnostic is stopped from the keyboard, control will go to the monitor instead of FLEX.

ADDRESS xxxx, DATA: nnnnnnnn
: An error was detected while testing the byte at address "xxxx". The data pattern read was "nnnnnnnn" (in binary). The data pattern written was an "all ones" pattern (11111111 binary).

TESTING COMPLETED
: The diagnostic has terminated, returning to the system.

REMARKS:

The 10 second delay is a software delay based on a 1 megahertz CPU clock. The delay itself is not critical, and no changes are required if a faster or slower CPU is used.

Program Name: QUICK
Program Type: MEMORY DIAGNOSTIC

PURPOSE:

QUICK performs a zeroes and ones check on a block of memory. This test is most frequently used as a quick check for solid failures.

Calling Sequence:

QUICK,starting-address,ending-address

where:

"starting-address" is the lowest address in the block to be tested.

"ending-address" is the highest address in the block to be tested. If none is specified, the FLEX "Memory End" value is used.

If no arguments are specified, the block from "0000" through the FLEX "Memory End" value is tested.

METHOD:

On each odd-numbered pass, QUICK zeroes out the block of memory being tested. Each byte is then complemented (making it hex FF) and checked. Any error is reported. The byte is then cleared and the next byte is processed.

On even-numbered passes, the block of memory being tested is set to an "all ones" pattern (hex FF). Each byte is cleared, checked, and reset to all ones with any error being reported.

The test runs until stopped by the user. After each pass, the keyboard is checked. If any character was entered, the test terminates immediately, returning to the operating system.

MESSAGES:

ERROR IN ADDRESS
An invalid hexadecimal character was detected in either "starting-address" or "ending-address". The test is aborted.

"QUICK" IS IN THE TEST AREA
The diagnostic "QUICK" resides in the block of memory being tested. The test is aborted.

STACK IS IN TEST AREA
The run-time stack used by QUICK is in the block of memory being tested. The test is aborted.

LAST < FIRST
"Ending-address" is less than "starting-address". The test is aborted.

FLEX ASSUMED OVERWRITTEN
The block of memory being tested contains all or part of the FLEX operating system. If the diagnostic is stopped from the keyboard, control will go to the monitor instead of FLEX.

ADDRESS xxxx, EXPECTED: nnnnnnnn, RECEIVED: nnnnnnnn
An error was detected while testing the byte at address "xxxx". The data pattern stored is that indicated by the EXPECTED value. The incorrect data pattern read back is that indicated by the RECEIVED value. Both values (nnnnnnnn) are given in binary.

TESTING COMPLETED
The diagnostic has terminated, returning to the system.

REMARKS:

QUICK takes approximately 0.64 seconds to test 4K (4096 bytes) of memory on a 1 MHz machine. Larger blocks take a proportionately longer time.

For parallel-organized memory, QUICK is a "walking zero" and "walking one" test that checks the eight parallel memory chips simultaneously. (See "Introduction to Memory Diagnostics" for a definition of "parallel-organized memory".)

Program Name: RANDOM
Program Type: MEMORY DIAGNOSTIC

PURPOSE:

RANDOM tests a block of memory using pseudo-random bit patterns as the test data.

Calling Sequence:

RANDOM,starting-address,ending-address

where:

"starting-address" is the lowest address in the block to be tested.

"ending-address" is the highest address in the block to be tested. If none is specified, the FLEX "Memory End" value is used.

If no arguments are specified, the block from "0000" through the FLEX "Memory End" value is tested.

METHOD:

On each pass, RANDOM fills the memory block being tested with data patterns generated by a pseudo-random number generator. The data in each memory location is then checked against the pattern written and discrepancies are reported. A fresh sequence of data patterns is used for each pass. The test runs until stopped by the user. After each byte is written or checked, the keyboard is interrogated. If a character has been entered, the diagnostic terminates, returning to the operating system.

MESSAGES:

ERROR IN ADDRESS
An invalid hexadecimal character was detected in either "starting-address" or "ending-address". The test is aborted.

"RANDOM" IS IN THE TEST AREA
The diagnostic "RANDOM" resides in the block of memory being tested. The test is aborted.

STACK IS IN TEST AREA
The run-time stack used by RANDOM is in the block of memory being tested. The test is aborted.

LAST < FIRST
"Ending-address" is less than "starting-address". The test is aborted.

FLEX ASSUMED OVERWRITTEN
The block of memory being tested contains all or part of the FLEX operating system. If the diagnostic is stopped from the keyboard, control will go to the monitor instead of FLEX.

ADDRESS xxxx, EXPECTED: nnnnnnnn, RECEIVED: nnnnnnnn
An error was detected while testing the byte at address "xxxx". The data pattern stored is that indicated by the EXPECTED value. The incorrect data pattern read back is that indicated by the RECEIVED value. Both values (nnnnnnnn) are given in binary.

TESTING COMPLETED
The diagnostic has terminated, returning to the system.

REMARKS:

RANDOM takes approximately 1.8 seconds to test 4K (4096 bytes) of memory on a 1 MHz machine. Larger blocks take a proportionately longer time.

The pseudo-random number generator in the diagnostic does not need to be initialized. RANDOM uses memory garbage as the seed, and forces the seed to be non-zero.

Program Name: WALK0
Program Type: MEMORY DIAGNOSTIC

PURPOSE:

WALK0 performs a "walking zero" test on a block of memory. At any time during the running of the test, only one bit in the entire block of memory being tested is a zero.

Calling Sequence:

WALK0,starting-address,ending-address

where:

"starting-address" is the lowest address in the block to be tested.

"ending-address" is the highest address in the block to be tested. If none is specified, the FLEX "Memory End" value is used.'

If no arguments are specified, the block from "0000" through the FLEX "Memory End" value is tested.

METHOD:

When invoked, WALK0 writes the entire block to be tested with an "all ones" pattern (hex FF). Starting at the lowest address, the right-most bit of the byte is set to zero and the byte rewritten. The byte is then checked. If correct, the bit that was cleared is set back to a one and the next bit in the byte is cleared and the byte rewritten. This process continues until all 8 bits are checked. If an error is detected, it is reported and the next bit is processed. The process then resumes with the next byte in memory. Thus, the cleared bit is "walked" through the entire block being tested. After the entire block has been tested, a plus sign is printed and the test repeats, starting with the lowest byte in the block. The test runs until stopped by the user. After each byte is tested, the keyboard is checked. If a character has been entered, the diagnostic terminates, returning to the operating system.

MESSAGES:

ERROR IN ADDRESS
An invalid hexadecimal character was detected in either "starting-address" or "ending-address". The test is aborted.

"WALKO" IS IN THE TEST AREA
The diagnostic "WALKO" resides in the block of memory being tested. The test is aborted.

STACK IS IN TEST AREA
The run-time stack used by WALKO is in the block of memory being tested. The test is aborted.

LAST < FIRST
"Ending-address" is less than "starting-address". The test is aborted.

FLEX ASSUMED OVERWRITTEN
The block of memory being tested contains all or part of the FLEX operating system. If the diagnostic is stopped from the keyboard, control will go to the monitor instead of FLEX.

ADDRESS xxxx, EXPECTED: nnnnnnnn, RECEIVED: nnnnnnnn
An error was detected while testing the byte at address "xxxx". The data pattern stored is that indicated by the EXPECTED value. The incorrect data pattern read back is that indicated by the RECEIVED value. Both values (nnnnnnnn) are given in binary.

TESTING COMPLETED
The diagnostic has terminated, returning to the system.

REMARKS:

WALKO takes approximately 3.5 seconds to test 4K (4096 bytes) of memory on a 1 MHz machine. Larger blocks take a proportionately longer time.

WALKO may be used regardless of the organization of memory. For parallel-organized memory, however, QUICK is a faster "walking bit" test. (See "Introduction to Memory Diagnostics" for a definition of parallel-organized memory".)

Program Name: WALK1
Program Type: MEMORY DIAGNOSTIC

PURPOSE:

WALK1 performs a "walking one" test on a block of memory. At any time during the running of the test, only one bit in the entire block of memory being tested is a one.

Calling Sequence:

WALK1,starting-address,ending-address

where:

"starting-address" is the lowest address in the block to be tested.

"ending-address" is the highest address in the block to be tested. If none is specified, the FLEX "Memory End" value is used.

If no arguments are specified, the block from "0000" through the FLEX "Memory End" value is tested.

METHOD:

When invoked, WALK1 writes the entire block to be tested with zeroes. Starting at the lowest address, the right-most bit of the byte is set to one and the byte rewritten. The byte is then checked. If correct, the bit that was set is cleared and the next bit in the byte is set to a one and the byte rewritten. This process continues until all 8 bits are checked. If an error is detected, it is reported and the next bit is processed. The process then resumes with the next byte in memory. Thus, the one bit is "walked" through the entire block being tested. After the entire block has been tested a plus sign is printed and the test repeats, starting with the lowest byte in the block. The test runs until stopped by the user. After each byte is tested, the keyboard is checked. If a character has been entered, the diagnostic terminates, returning to the operating system.

MESSAGES:

ERROR IN ADDRESS
An invalid hexadecimal character was detected in either "starting-address" or "ending-address". The test is aborted.

"WALK1" IS IN THE TEST AREA
The diagnostic "WALK1" resides in the block of memory being tested. The test is aborted.

STACK IS IN TEST AREA
The run-time stack used by WALK1 is in the block of memory being tested. The test is aborted.

LAST < FIRST
"Ending-address" is less than "starting-address". The test is aborted.

FLEX ASSUMED OVERWRITTEN
The block of memory being tested contains all or part of the FLEX operating system. If the diagnostic is stopped from the keyboard, control will go to the monitor instead of FLEX.

ADDRESS xxxx, EXPECTED: nnnnnnnn, RECEIVED: nnnnnnnn
An error was detected while testing the byte at address "xxxx". The data pattern stored is that indicated by the EXPECTED value. The incorrect data pattern read back is that indicated by the RECEIVED value. Both values (nnnnnnnn) are given in binary.

TESTING COMPLETED
The diagnostic has terminated, returning to the system.

REMARKS:

WALK1 takes approximately 3.5 seconds to test 4K (4096 bytes) of memory on a 1 MHz machine. Larger blocks take a proportionately longer time.

WALK1 may be used regardless of the organization of memory. For parallel-organized memory, however, QUICK is a faster "walking bit" test. (See "Introduction to Memory Diagnostics" for a definition of parallel-organized memory".)

INTRODUCTION TO DISK DIAGNOSTICS AND UTILITIES

INTRODUCTION TO DISK DIAGNOSTICS AND UTILITIES

The diskette diagnostics and utilities in this package provide a mechanism for detecting errors on a FLEX™ formatted disk and, in some cases, retrieving the damaged or lost data. The programs in this package will not perform miracles, however. As with any set of diagnostics, there are probably some problems that they will not be able to detect. It is hoped that these programs will be able to detect the more common problems, and provide enough warning to the user so that some of the data affected may be salvaged. The user should be aware, however, that there are some types of failures from which data cannot be recovered.

This introduction will discuss the structure of a FLEX disk and some of the problems that may show up. Following this introduction is documentation on each of the programs. Lastly, several case studies are analyzed to give a feeling as to how to use the various programs in this package.

Disks in General

The most common di.sk storage available on microcomputer systems is the "floppy disk". These are also called "flexible disks" because they are thin and will bend easily. There are also disks called "hard disks" which contain rigid plates. Hard disks are usually quite expensive and can hold large amounts of data.

A disk is usually organized into "tracks" and "sectors". The tracks may be compared to the rings in an archery target in that they are concentric circles on the disk. The number of tracks on a disk varies with the size of the disk; smaller disks usually have fewer tracks. Tracks are numbered sequentially, starting at 0. Each track is divided into sections called "sectors". The sectors are the areas in which data is stored. The number of-sectors in a track depends on the number of bytes that are going to be stored in each sector; the more bytes that are stored in a sector, the fewer sectors that will fit in a track.

Information is written to or read from the disk by a read/write head in the disk drive. The head is at the end of an arm which positions it over the proper track. As the disk spins, the sectors pass under the head of the disk drive. To read or write information, the disk drive waits for the right sector to come under the head, then it reads or writes the data into the sector.

The method that the disk drive uses to determine when the right sector is under the head depends on the disk and the drive. Some disks are "soft-sectored", others are "hard-sectored". "Hard-sectored" diskettes have holes punched in them, one corresponding to the beginning of each sector. A separate hole, called the "index mark" indicates the beginning of the track. "Soft-sectored" diskettes have the sector

---

FLEX is a trademark of Technical Systems Consultants, Inc.

number, and other information, recorded magnetically in an address field at the front of each sector. The disk drive reads each sector until it finds the one that it wants. There is usually only one hole in a soft-sectored disk; the one that marks the start of a track.

It is important to note that if the information that is written in front of the sector is destroyed, the disk drive will not be able to find the sector. Some disk controllers are smart enough to detect this case, others are not. Those that do detect it return an error that indicates that the sector could not be located. Those controllers that cannot detect such a condition will continue to look for the sector until the machine is reset.

All of the information that is written to a disk is checksummed, and the checksum is written on the disk right after the information. When a sector is read, the data that comes off of the disk is checksummed again. If the checksum just computed does not match that which was recorded on the disk, a checksum error (also called a CRC error) exists. Checksum errors indicate that the data was not written correctly, or can not be read correctly.

Structure of a FLEX Disk

Some information about the structure of a FLEX disk is given in the "FLEX Advanced Programmer's Guide". This information is summarized below.

A FLEX disk is given its structure by the initialization program, NEWDISK. This program writes the address fields in front of the sectors of the diskette and defines the size of the sectors (256 bytes). Sectors in each track are numbered starting at 1.

The sectors on a FLEX diskette are often indicated by a "disk address", also called a "sector address". This address is a hexadecimal number consisting of the number of the track containing the sector concatenated with the number of that sector within the track. For example, the disk address 030A means "track 03, sector 0A".

After initialization, the sectors on all of the tracks, except the first track, are linked together into a chain. This is accomplished by reserving the first two bytes of each sector as a "link field" that contains a pointer to another sector. The next two bytes are also reserved. They are used to hold a record number when the sector is part of a file.

The first track on a FLEX disk, track 0, is special. This track contains the boot sectors, the system information sector, and the directory. The boot sectors contain a small program that fs read into memory when the system is brought up. The boot sectors are different from all of the other sectors in that they do not have a link field or a record number field. On a standard FLEX disk, the boot sectors are sectors 1 and 2 on track 0.

The system information sector is in sector 3 of track 0. This sector contains the name of the diskette, number of the diskette, and the date that the diskette was initialized. Also in the system information sector are the sector addresses of the beginning and end of the free chain and the number of sectors in the free chain. Lastly, the largest value for a track number and the largest value for a sector number within a track are stored here for reference purposes. The actual locations of this information in the sector is given in the following table:

Byte Number	Content
00-15	Zeroes
16-26	Disk Name
27-28	Volume Number
29-30	Start of free chain
31-32	End of free chain
33-34	Size of free chain
35-37	Initialization Date
38	Maximum Track Number
39	Maximum Sector Number
40-255	Reserved for future expansion

Track 0 sector 4 is not used by FLEX, being reserved for future expansion. The disk directory starts at sector 5 on track 0, and initially is composed of the rest of the sectors on track 0. The directory sectors are linked together, the same as in the free chain. There are no record numbers in the directory, however. A directory sector has the following format:

Byte Number	Content
00-01	Sector link
02-15	Zeroes
16-39	Directory Entry
40-63	Directory Entry
64-87	Directory Entry
88-111	Directory Entry
112-135	Directory Entry
136-159	Directory Entry
160-183	Directory Entry
184-207	Directory Entry
208-231	Directory Entry
232-255	Directory Entry

The description of an individual directory entry is given in the FLEX Advanced Programmer's Guide.

Once a sector becomes part of a file, the sector link field contains a pointer to the next sector in the file. If the link is zero, the sector is the last sector in the file. The sectors in a file are numbered starting at 1. This number is kept in the record number field of the sector (bytes 2-3). The remaining 252 bytes (4-255) contain data.

Random Files

A FLEX random file is structured like an ordinary sequential file. Each sector contains a sector link and a record number, plus 252 bytes of data. The difference is that there are two sectors in front of the first data sector that contain a compressed ordered list of the sectors in the file. These two sectors are called the "file sector map". By looking at the sector map, FLEX can quickly determine which sector contains which record. The two map sectors themselves have record numbers of zero.

The information in the map sectors is composed of three-byte fields. In each field, the first two bytes are a disk address (track and sector), and the next byte is a sector count byte. The count byte specifies the number of sectors in the file starting at that disk address, that are in sequence. If all three bytes are zero, it indicates that the end of the map has been reached. How the sector map works may be best explained through the use of an example. Assume that the sector map of a file looks like this:

```
02 06 03
0F 06 01
03 0E 05
00 00 00
```

The leftmost two bytes in each of the above lines are the disk address, and the rightmost byte is the sector count. The first line, 02 06 03 means: "starting at disk address 0206, there are 3 sectors that are in the file". These would be sectors 0206, 0207, and 0208. The next line says that starting at 0F06 there is one sector that is in the file. This, of course, would be 0F06 itself. The third line says that there are 5 sectors starting at 030E. If we assume that there are 15 sectors in a track (01 through 0F), then these sectors would be 030E, 030F, 0401, 0402, and 0403. The last line, being all zeroes, indicates that the end of the file has been reached. Summing this up, the file occupies the sectors 0206, 0207, 0208, 0F06, 030E, 030F, 0401, 0402, and 0403, in that order. Thus, if we want sector 6 in the file, we would go to sector 030F.

To create a random file, it is necessary to set a flag in the file control block immediately after opening the file for writing, and before writing any data to it. The exact procedure is described in the FLEX Advanced Programmer's Guide.

How FLEX Handles Files

After a disk has been initialized, all of the sectors are linked together in the free chain. When a file is created, sectors are removed from the head of the free chain and assigned to the file as they are needed. When a file is deleted, the chain of sectors that were in the file is concatenated to the end of the free chain.

As mentioned earlier, the system information sector contains the disk addresses of the beginning and end of the free chain, as well as its size. The information in this sector is not updated each and every time a sector is removed from the free chain and assigned to a file. doing this would result in a lot of overhead when files are being written. Instead, this information is kept in memory and updated there. The system information sector on the disk is updated when FLEX returns th command mode and issues the plus signs for a prompt. it is also updated whenever the FMS "close all files" routine is called. For this reason, one should not change a disk in the middle of a program, even if the files that were being written have been closed, since the free chain information has not been updated on the disk.

Types of Disk Problems

There are three broad classes of disk problems: hardware-caused, software-caused, and human-caused. Hardware-caused problems may result in the inability to read of write a disk, or the changing of the information on the disk. Software-caused problems may result in the changing of information, but very rarely do they result in the inability to read of write a disk. (Of course, we are assuming that the software that drives the disk is working correctly.) Human-caused problems are the result of the user doing something that is ill-advised, such as resetting the machine while a file is being written. Such an action could cause the integrity of the disk to be compromised. The links in the file chains may be wrong; the directory information about a file may be inaccurate; of the free chain may disappear.

The inability to read or write a disk may be caused by the disk itself, the disk drive, or the disk controller. Isolating the problem is a simple process of elimination. If only one disk can not be read, then the disk is probably at fault. If the disk will work in one drive, but not another, then the drive may have a problem. If none of the disks and drives work, then the problem is probably in the controller or cable from the controller to the disk drives.

If only a few sectors on a disk cannot be read or written, then the disk itself has some bad spots. Modern disk manufacturing techniques produce high quality media, but it is possible for some bad spots to appear. floppy disks are prone to wear, since the disk drive head actually is in contact with the diskette. In hard disks, however, the head rides a cushion of air above the surface of the disk, so that the disk surfaces experience no wear. The majority of failures in hard disk systems are electronic failures.

The most common problems with disk is that of bad sectors, those that get checksum errors. Normally, FLEX detects bad sectors when the disk is initialized, at which time they are removed from the free chain and not made available for use in a file. Sometimes, however, a bad spot may be pattern sensitive. Such a bad spot may appear to be a good sector during initialization, but gets a checksum error when used in a file. It is also possible for a sector to become bad due to physical damage to the disk or due to wear. Handling the various types of errors

is discussed further in the Case Studies section of this manual.

Software-caused problems usually involve the alteration or destruction of data in a file. Such problems are sometimes the result of programming errors, especially in assembly language programs that read and write files. It is good practice to test such programs on scratch disks so that no valuable data is lost if a programming error destroys the data on the disk. Sometimes the directory, free chain, or the file chains may be destroyed. This type of problem is called a "structural" problem, since the normal structure of a FLEX disk has been damaged. Recovering most of the data from structurally damaged disks is usually possible. However, the structural damage usually cannot be easily repaired. The best way to recover from structural damage is to copy the data to another disk and re-initialize the bad one. Handling structural errors is also discussed in the Case Studies section of this manual.

Human-caused problems are usually structural problems. Most-of the problems occur when something is done while a file is being written, or is open for writing. Resetting the machine, opening the disk drive door, or removing the diskette at the wrong time can result in structural damage. Sometimes, however, there is no choice. If a program is running wild, or the user suddenly realizes that the program is writing on the wrong disk, there may be no other alternative than to take drastic action to stop the program. If such action is taken, the next step should be to run some tests on the diskette to make sure that no damage was done. Some structural problems do not show up until long after they were caused.

THE DISK UTILITIES IN THIS PACKAGE

There are 10 programs in this package for use in isolating and recovering from problems. In this section, we will quickly look at each of them.

Three programs (TEST, VALIDATE, and FILETEST) are diagnostics which check for bad spots and structural problems. TEST is a fast checkout program for detecting bad spots. FILETEST also looks for bad spots, but it can tell you in which file the bad spot is located; TEST cannot. However, FILETEST runs more slowly than TEST. VALIDATE checks for structural problems, such as intersecting files or discrepancies between the file and the information about the file that is in the directory.

Four programs (RAWCOPY, REBUILD, RECOVER, and UNDELETE) are data recovery utilities. Two of these, REBUILD and RECOVER, are for recovering data from a disk when the directory has been destroyed. REBUILD attempts to recover all of the files on the disk, while RECOVER will retrieve only selected files. RECOVER does require that the user know the starting track and sector of the file. RAWCOPY will copy a file that has a checksum error in it. The data in the bad sector will be damaged, but it is assumed that the user can repair the damage once a readable copy is made. UNDELETE can recover files from the free chain, assuming that they have not been overwritten by new files.

Of the remaining three programs, COPYR is used to restore the file sector map to a random file after the file has been recovered by REBUILD. It can also be used to put a file sector map on any sequential file. FLAW is used to remove bad sectors from the free chains. This is particularly useful when a sector is intermittent; that is, it is not detected by NEWDISK, but it gives errors when it is used in a file. Lastly, EXAMINE is a general read/write/modify utility that can operate on individual sectors. With this program, it is possible to change any sector on a disk.

Knowing when to use the various programs will cone with experience. The Case Studies section of this manual gives examples on how the various programs can be used with each other to isolate and correct problems.

System Dependencies

All of the disk diagnostics and utilities in this package are written to run under the FLEX Operating System. It is not possible to run these programs under other operating systems.

Some the the programs read the system information sector to get information about the disk. In particular, they want to know the number of tracks on the disk and the number of sectors in a track. In order to protect against the possiblity of working with bad information, the values that are read from the system information sector are compared against a table of acceptable values.

The table consists of several two-byte entries. The first byte is the number of the last track on the disk, and the second byte is the number of the last sector in a track. For double density diskettes, this is the number of the last sector in a data track, not the directory track. (The directory track on a double density diskette is written in single density.) The following is a description of the entries in the table.

Largest Track	Largest Sector	Description
39	10	5" Single-sided, 40 tracks
39	20	5" Double-sided, 40 tracks
34	10	5" Single-sided, 35 tracks
34	20	5" Double-sided, 35 tracks
76	15	8" Single-sided, single-density
76	26	8" Single-sided, double-density
76	30	8" Double-sided, single-density
76	52	8" Double-sided, double-density
255	255	Hard Disk

The table is terminated by three zero bytes, the first two of which are available for the user to add information about a non-standard configuration. The last zero byte should not be changed. This table is located at location 0003 in EXAMINE and RAWCOPY, and at location C103 in TEST and REBUILD. The other diagnostics do not have this table.

If the maximum track and sector values do not correspond to one of the entries in the table, it is assumed that the data in the system information sector is incorrect. When this happens, the program prompts the user for information about the disk. The program will also prompt for information if the system information sector cannot be read because of a disk error. The following is a description of the prompts.

MAXIMUM TRACK/SECTOR READ: tt/ss ARE THESE ACCEPTABLE? (Y/N)
: The values read from the system information sector were "tt" and "ss". There was no corresponding entry in the table. If these are the correct values, type "Y"; if not, type "N". If "Y" is typed, these values will be accepted as correct. If "N" is typed, additional prompts will follow.

HARD DISK OR FLOPPY DISK (H/F)
: If the disk being tested is a hard disk, type "H"; if it is a floppy disk, type "F". If it is a hard disk, the maximum values will be set to 255 each. If it is a floppy disk, additional prompts will follow.

DISKETTE SIZE (5/8)
: If the diskette is an 8 inch diskette, type "8"; if a 5 1/4 inch diskette, type "5".

35 TRACKS OR 40 TRACKS (3/4)
If the diskette is a 5 1/4 inch diskette and it was formatted for 35 tracks, type "3". If it was formatted for 40 tracks, type "4".

SINGLE OR DOUBLE SIDED (S/D)
If the diskette is a single-sided diskette, type "S"; if it is a double-sided diskette, type "D".

SINGLE OR DOUBLE DENSITY (S/D).
If the diskette is a single-density diskette, type "S"; if it is a double-density diskette, type "D". This prompt appears only if the diskette is an 8 inch diskette.

If the system information sector could not be read because of a disk error, prompting beings with the HARD DISK OR FLOPPY DISK message.

DISK TROUBLESHOOTING GUIDE

Introduction

This portion of the manual gives hints on using the disk diagnostic and repair programs in this package. With experience, you will become familiar with the capabilities and limitations of each program, and will be better able to judge which program will work best for a particular problem.

Symptoms of a Problem

Most of the time, you will know that there is a problem because of an error message issued by FLEX. The messages that are sure indicators of a problem are:

```
DISK FILE READ ERROR
DISK FILE WRITE ERROR
RECORD NUMBER MATCH ERROR - FILE DAMAGED
```

In some cases, an unexpected error message may be a clue. For example, a DRIVES NOT READY message when, indeed, they are ready, may be an indication that the disk controller could not find a sector. On 5 1/4 inch disk drives, some controllers cannot detect a NOT READY or SECTOR NOT FOUND condition, so if the drive apparently "hangs up" with the head loaded, it may also be an indication that the controller cannot find a sector.

A clue that there may be a structural problem with a disk is an unexpected ALL AVAILABLE DISK SPACE HAS BEEN USED message. If there was a lot of space on the disk, and suddenly it all vanishes, then the free chain has been destroyed. This is a warning that the structural integrity of the disk should be checked.

Here is a partial list of some other events which should be taken as warnings that a disk may be damaged.

1) Data changing in a file.
   If a file that has not been re-written suddenly has different data in it, it might indicate that another file is linked to it. This is a severe structural problem that could result in a loss of all of the data on the disk.

2) A file disagrees with its directory entry.
   If a file is obviously much larger or much smaller than the size that is in the directory, then something is wrong. It might be that the file was truncated, or linked into the free chain.

3) Duplicated or missing names in the directory.
   It is not possible to create a file with the same name as an existing file. If there are two files with the same name, then the directory has been damaged.

4) Memory problems in the machine.
   If a machine has recently had a memory problem, then all disks should be checked once the memory problem is fixed. A memory problem can cause programs, including FLEX, to run wild. Runaway programs can cause structural damage to the data on a disk. Such damage may not be immediately apparent.

Most of the diagnostics do not take a long time to run. So, a good general rule is to run diagnostics if there is any doubt about the physical or structural integrity of a disk.

General Hints

As mentioned before, experience is the best teacher as to which diagnostics are most suitable for any given problem. However, there are some general guidelines.

1) If you get a disk error while reading a file, use FILETEST with the "A" option to determine which file contains the error. The next step is to try reading the disk on another disk drive. If the error is "soft", the other drive might be able to read the file. If so, copy the file immediately.

2) If you get an error on a disk, you should first decide if it is worth the effort to try to salvage what is left of the file. If the data can be regenerated easily from backups or previous versions, it might be faster to do that than to try to recover the damaged data.

3) VALIDATE is the preferred test for detecting structural damage.

4) Severe structural damage, such as files being linked together, is very difficult to repair. To do so requires a good knowledge of how files are contructed on a FLEX disk. In many cases, the best course is to copy as much as possible to another disk and re-initialize the crashed disk. An editor can then be used to salvage as much as possible on the good disk.

5) If the directory is damaged and cannot be read, the only hope is to use REBUILD. RECOVER can be used for individual files if the starting track and sector of each file is known.

DISK DIAGNOSTICS AND UTILITIES DESCRIPTIONS

Program Name: COPYR
Program Type: DISKETTE UTILITY

PURPOSE:

COPYR copies a file, forcing the new copy to be a random file. It is intended to be used after a file has been recovered with the REBUILD utility, which cannot recover the file sector map.

Calling Sequence:

COPYR old-file,new-file

where:

"old-file" is the file specification of the file that is to be copied. The default extension is ".SCR".

"new-file" is the specification of the file that is to be written. This file will be the random file. The default extension is ".SCR".

METHOD:

COPYR performs a simple file copy function, declaring the new file to be a random file. The FLEX operating system is used for all system interfacing. No special handling of errors is performed. COPYR assumes that the file is not damaged and that all sectors are readable.

MESSAGES:

COPY COMPLETED
The copy operation has terminated normally.

All other messages are produced by the FLEX operating system.

REMARKS:

COPYR is used primarily to restore the file sector map portion of random files which have been recovered by the REBUILD utility. Any file may be copied with COPYR, and the resulting copy will be a random file. Note, however, that some programs that use random files, such as BASIC, require that the data be organized in a special way in each sector. If COPYR is used to make an arbitrary sequential file a random file, the random file may not conform to the requirements of the program that uses it. Only those files that were originally random files, and which have lost their file sector map, should be made random with COPYR.

Program Name: EXAMINE
Program Type: REPAIR UTILITY

PURPOSE:

EXAMINE is a repair utility which allows the user to read, modify, or write any sector on a FLEX diskette.

Calling Sequence:

EXAMINE drive-number

where:

"drive number" is the drive containing the diskette to be examined. The diskette must already be mounted. If no drive number is specified, the work drive is used if it has not been set to "all". If the work drive is "all", a drive must be specified.

METHOD:

EXAMINE starts by trying to read the System Information Sector from the specified drive. If successful, it determines the configuration (diskette size, number of sides, and density) from that information. If the System Information Sector cannot be read, EXAMINE will prompt for the information necessary to determine the configuration. Once the configuration is known, EXAMINE is ready to accept commands.

Commands:

EXAMINE indicates that it is ready for a command by issuing the prompt:

COMMAND:

There are nine valid commands. Each command consists of a single letter, optionally followed by a parameter. The valid commands are:

R,<sector address>	Read a sector
W,<sector address>	Write a sector
D,<sector address>	Read and display a sector
C,<sector address>	Read and display until end of file
M,<byte number>	Modify sector buffer
F,<file spec>	Read first sector of a file
B,<flle spec>	Build link table for a file
T,<addr>,<addr>,<count>	Move data in memory
S	Stop, return to FLEX

The parameter <sector address> will be described later on.

R - Read a Sector into the Internal Sector Buffer

The sector specified as the parameter is read into a sector buffer internal to EXAMINE. If an error was detected during the reading of the sector, the appropriate error message is printed. If the error was SECTOR NOT FOUND or DRIVE NOT READY, then no data was transferred to the internal sector buffer.

D - Read and Print (Dump) a Sector.

This command reads the specified sector into the internal sector buffer, then prints its content at the terminal. If a SECTOR NOT FOUND or DRIVE NOT READY error is detected, no information is read or printed.

W - Write the Internal Sector Buffer to a Sector

The content of internal sector buffer is written to the sector specified by the argument to the command. If "verify" is "on", then the sector will be read after having been written. Any error detected during the write or during a subsequent verification is reported. If the error is SECTOR NOT FOUND or DRIVE NOT READY, then no data was written to the diskette.

C - Print a Chain of Sectors.

This command reads and prints the content of a chain of sectors as defined by the sector links. The information is printed in the same form as the "D" command. The argument to this command is the address of the sector at which to start the dumping. The dumping stops when the end of the chain is reached, or a SECTOR NOT FOUND or DRIVE NOT READY error is encountered. As with any output from FLEX, the printing may be stopped at any time by using the escape key. Typing an escape followed by a carriage return will stop the printing and a new command will be requested.

M - Modify Internal Sector Buffer Data

This command allows the examination and changing of the data in the internal sector buffer. The internal sector buffer must have been previously loaded with data by the "R" or "D" command. The argument to this command is the number of the byte, in hexadecimal, at which to start the examination. If no argument is specified, byte zero is assumed. When invoked, the byte number and its current content are displayed. Typing a two-digit hexadecimal number will cause the content to be changed to that number. Typing an up arrow (a circumflex on some keyboards) will cause the previous byte to be displayed. Typing a carriage return will return to command mode. Typing any other separator character, such as a period, will cause the next byte to be displayed. The display is circular; byte 00 is considered to follow byte FF. This command does not update the sector on the diskette; it only changes the data in the buffer. To update the data on the diskette, the "W" command must be used, after modification of the buffer, to write the updated internal sector buffer to the desired sector.

F - Read First Sector of File

The argument to this command is a FLEX file specification. The default extension is ".TXT". The diskette directory is searched for the file and, if found, the first sector of the file is read into the internal sector buffer.

There are four names which may be specified instead of a legitimate file specification to cause the first sector of special areas of a FLEX diskette to be read. This is a convenience so that the specific sector addresses do not have to be memorized or determined from other data. These names and the areas to which they refer are:

$B -- The boot sector
$S -- The system information sector
$D -- The directory chain
$F -- The free chain

For example, the command "F,$D" will cause the first sector of the directory to be read into the internal sector buffer. If desired, the entire directory may then be dumped by typing "C".

B - Build Link Table for File.

The argument to this command is a FLEX file specification or one of the special names described under the "F" command. If no argument is specified, the current sector address is assumed to be the start of the chain. When invoked, this command reads the chain, storing the link from each sector in a table in memory. After the table is built, the first sector is re-read and becomes the current sector. This table is used whenever a "P" is specified as a disk address. Thus, once the table is built, one may move both forwards and backwards along that chain by specifying disk addresses of "N" and "P". Each invocation of the "B" command erases the previous content of the link table in memory. Thus, only one file chain at a time can occupy the link table.

T - Transfer (Move) Data in Memory.

This command allows the moving of data in memory. It has three arguments, an address indicating where the data is currently located, an address indicating where the data is to be moved, and a count of the number of bytes to move. The two addresses are in hexadecimal, and the count is in decimal. There are two special forms of address which refer to the information in the internal data buffer. These are *B and *D. *B represents the address of the internal data buffer. *D represents the address of the data portion of the internal sector buffer (4 bytes beyond *B). If either *B or *D is used as an address, then the count parameter is optional. If *B is specified add no count is specified, 256 bytes will be copied. If *D is specified without a count, 252 bytes will be copied. A count field may be used to move fewer bytes. Moving more that 256 bytes into the internal b@ffer area will produce unpredictable results. Care must also be taken so that data is not moved on top of EXAMINE. EXAMINE and its buffers start at loction OOOO and use up about 4K bytes; so data should not be moved to an

area below $1000. Following are examples of the use of this command.

T,3000,4000,100
Move 100 bytes from $3000 to $4000.

T,*B,5000
Copy the content of the internal sector buffer to address $5000. All 256 bytes will be copied.

T,4020,*D,20
Copy twenty bytes of information from address $4020 to the data area of the internal sector buffer. The link field and record number field of the sector buffer are not changed.

S - Return to FLEX

Specifying Sector Addresses

EXAMINE always has a "current sector address", specified as a single hexadecimal number. This is the address of that sector (called the "current sector"), which is to be read or written. A sector address is of the form "ttss", where "tt" is the track number, and "ss" is the sector number within the track. The current sector address starts out at 0003 (track 00, sector 03), which is the address of the System Information Sector. When a command which has a sector address as an argument is typed, the argument becomes the current sector address. If an illegal address is entered, an error message is given and the current sector address does not change. If no sector address is specified to a command that accepts one, then the action is performed on the current sector.

A sector address in a command may be specified in one of several ways. The simplest, and most useful, forms are:

ttss Specify Track and Sector Explicitly.
In this form, "tt" is the track number in hexadecimal, and "ss" is the sector number in hexadecimal. The operation appropriate to the command is performed on the explicitly specified track and sector. For example, 010D is track 01, sector 0D.

\+ Next Physical Sector.
Specifying a plus sign as the sector address causes the current sector address to be incremented by 1. The resulting value becomes the new current sector address, and the command acts upon that address. If the current sector address is the last physical sector on a track, the the new current sector address is the first sector of the next track. If the current sector address is the last physical sector on the diskette, the new current sector address is the first sector of the first track on the diskette. For example, if the current sector address is 0104, then specifying a plus sign as the argument to a command

will cause the command to act on the sector with the sector address 0105.

- Previous Physical Sector.
  Specifying a minus sign as the sector address causes the current sector address to be decremented by 1. The resulting value becomes the new current sector address, and the command acts upon that address. If the current sector address is the first physical sector on a track, the new current sector address is the last sector of the previous track. If the current sector address is the first physical sector on the diskette, the new current sector address is the last sector of the last track on the diskette. For example, if the current sector address is 0105, then specifying a minus sign as the,argument to a command will cause the command to act on the sector with the sector address 0104.

N Next Logical Sector
  If an "N" is specified as the sector address, the sector at the current sector address is read, and its link becomes the new current sector address. The command then acts on this new sector address. The "W" command does not accept this form of sector address. This form of sector address allows one to step through a file on a diskette. If the link in the current sector is zero (an end of file), then the message END OF CHAIN is issued and the command does nothing. For example, if the current sector is 0104 and its link points to sector 0301, then 0301 becomes the current sector address, and the command acts on that sector.

P Previous Logical Sector
  This form of sector address requires the "B" command to have been typed at some time. If a "P" is specified as the argument to a command, the sector link table, built by the "B" command, is searched for the current sector address. If it is found, then the sector which points to the current sector becomes the new current sector, and the command acts on it. If the current sector is not in the link table, or is the first sector in the link table, then a message is issued and the command does nothing. The "W" command does not accept this form of sector address.

= Use Current Sector
  If an equal sign is specified as the argument to a command, then the current sector address does not change. This is equivalent to not specifying any parameter to the command.

The above forms of sector address are the simplest forms. There are more complex forms which result in greater flexibility in specifying sector addresses as parameters to commands. The symbols "+", "-", and "=" may be combined with each other and with track or sector numbers to form a sector address. For example, 3=, +4 and =+ are all legal forms. In these forms, the item to the left refers to the track, and the item

to the right refers to the sector. Thus, 3= means "set the track to 3 and do not change the sector"; +4 means "increment the track number and set the sector number to 4"; and =+ means "do not change the track and increment the sector". A table at the end of the documentation for this program lists all of the legal forms and their effects on the current sector address.

MESSAGES:

ADDRESS: ttss, CRC ERROR
A CRC error (Checksum Error) was detected by the disk controller when reading the sector at sector address "ttss".

ADDRESS: ttss, DRIVE NOT READY
An attempt was made to read or write the sector at sector address "ttss", but the controller indicated that the drive was not ready.

ADDRESS: ttss, SECTOR NOT FOUND
An attempt was made to read or write the sector at sector address "ttss", but the controller could not locate the sector on the diskette. This usually indicates that the sector address field on the diskette has been destroyed.

BAD LINK TO NEXT SECTOR
A command which follows links in a file chain encountered a link which specified a track not on the diskette, a sector number of zero, or a sector number larger than the maximum on the track.

COMMAND:
The prompt for the next command.

DRIVE MUST BE SPECIFIED
No parameter was specified when EXAMINE was called, and the working drive was set to "ALL". If the working drive is "ALL", a drive number must be specified as a parameter when calling EXAMINE.

END OF CHAIN
A command that normally follows the links in a file chain reached the end of the chain. These commands include "B" and "C". Also, any command invoked with the logical sector address "N" (Next Logical Sector), may give this message if the current sector is an end of file.

ILLEGAL ADDRESS SPECIFIED
A memory address to the "T" command was not a valid-hexadecimal number or *B or *D, or the sector address specified as the argument to a command was not one of the legal forms. See the table at the end of the documentation for this program for a summary of the legal forms.

ILLEGAL COUNT SPECIFIED
An illegal decimal number was specified as the count parameter to the "T" command.

ILLEGAL DRIVE NUMBER
An invalid drive number was specified as the parameter to EXAMINE.

INVALID BYTE NUMBER
An illegal hexadecimal number was entered as the argument to the "M" command, or the number entered was greater than $FF. Note that because FLEX is used to assemble the byte number, lower case hexadecimal digits are not allowed.

NO PREDECESSOR FOUND
A command was invoked with an argument of "P" (Previous Logical Sector), but the current sector was the first sector in the link table. Thus, the current sector is the first sector in the chain that was scanned when the table was built.

SECTOR NOT IN LINK TABLE
A command was invoked with an argument of "P" (Previous Logical Sector), but the current sector was not in the sector link table. Either the "B" command was not previously invoked to build the link table, or the current sector is not a part of the file chain which was scanned when building the table.

SYSTEM INFO SECTOR INVALID
The diagnostic could not read the system information sector on the diskette, or the information read concerning maximum track and sector did not appear correct. This message will be followed by prompts for disk configuration information. See "The Disk Utilities in This Package: System Dependencies" for details.

UNKNOWN COMMAND
The command entered could not be recognized.

WRITE ON LOGICAL SECTOR NOT ALLOWED
The "W" command (Write Sector) may not have "P" or "N" as an argument.

Legal Forms of Sector Address

SECTOR ADDRESS FORM	EFFECT ON TRACK NUMBER (See Note 1)	EFFECT ON SECTOR NUMBER (See Note 2)
ttss	Set to "tt"	Set to "ss"
tt=	Set to "tt"	Unchanged
tt+	Set to "tt"	Incremented
tt-	Set to "tt"	Decremented
=ss	Unchanged	Set to "ss"
==	Unchanged	Unchanged
=+	Unchanged	Incremented
=-	Unchanged	Decremented
+ss	Incremented	Set to "ss"
+=	Incremented	Unchanged
++	Incremented	Incremented
+-	Incremented	Decremented
-ss	Decremented	Set to "ss"
-=	Decremented	Unchanged
-+	Decremented	Incremented
--	Decremented	Decremented
+	(See Note 3)	Incremented
-	(See Note 4)	Decremented
=	Unchanged	Unchanged
(Return)	Unchanged	Unchanged

1) If the effect is to increment the track number, and the current sector is on the last track of the diskette, then the track number is set to zero. If the effect is to decrement the track number, and the current sector is in track zero, then the track number is set to the last track on the diskette.

2) If the effect is to increment the sector number, and the current sector is the last sector in the track, then the sector number is set to one. If the effect is to decrement the sector number, and the current sector is the first sector on the track, then the sector number is set to the last sector of the track.

3) The track number will be incremented if the current sector is the last sector in the track.

4) The track number will be decremented if the current sector is the first sector in the track.

EXAMPLE

As an example of the use of some of the features of EXAMINE, let us assume that we have a diskette in which a file name in the diskette directory has been damaged. Let us further assume that the file name is supposed to be NEWDISK.CMD, but that one of the letters has been somehow changed to a control character. We would like to change the bad character to that which it should be.

The first step is to put the damaged diskette in the work drive and type EXAMINE. The following is an annotated example of how the session might go.

```
COMMAND: F,$D                          <Read first sector of directory>
COMMAND: D                             <Dump the first sector>

DISK ADDRESS: 0005
   -0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -A -B -C -D -E -F
0- 00 06 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
1- 45 52 52 4F 52 53 00 00 53 59 53 00 00 01 01 01 ERRORS__SYS______
2- 09 00 09 02 00 01 04 50 46 4C 45 58 00 00 00 00 _______PFLEX_____
3- 53 59 53 00 00 01 0A 03 04 00 19 00 00 01 04 50 SYS_____________P
4- 50 52 49 4E 54 00 00 00 53 59 53 00 00 03 05 03 PRINT___SYS______
5- 05 00 01 00 00 01 04 50 43 41 54 00 00 00 00 00 _______PCAT______
6- 43 4D 44 00 00 03 06 03 08 00 03 00 00 01 04 50 CMD_____________P
7- 43 4F 50 59 00 00 00 00 43 4D 44 00 00 19 0F 1D COPY____CMD______
8- 04 00 05 00 00 01 0A 50 44 45 4C 45 54 45 00 00 _______PDELETE___
9- 43 4D 44 00 00 03 0E 03 0F 00 02 00 00 01 04 50 CMD_____________P
A- 4C 49 53 54 00 00 00 00 43 4D 44 00 00 04 01 04 LIST____CMD______
B- 03 00 03 00 00 01 04 50 50 00 00 00 00 00 00 00 _______PP________
C- 43 4D 44 00 00 04 04 04 04 00 01 00 00 01 04 50 CMD_____________P
D- 41 53 4E 00 00 00 00 00 43 4D 44 00 00 04 05 04 ASN_____CMD______
E- 05 00 01 00 00 01 04 50 52 45 4E 41 4D 45 00 00 _______PRENAME___
F- 43 4D 44 00 00 04 06 04 06 00 01 00 00 01 04 50 CMD_____________P

                                       <Not in this sector>
```

(continued)

```
COMMAND: D,N                              <Dump the next sector>

DISK ADDRESS: 0006
   -0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -A -B -C -D -E -F
0- 00 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
1- 41 50 50 45 4E 44 00 00 43 4D 44 00 00 04 07 04 APPEND__CMD______
2- 09 00 03 00 00 01 04 50 42 55 49 4C 44 00 00 00 _______PBUILD____
3- 43 4D 44 00 00 04 0A 04 0A 00 01 00 00 01 04 50 CMD_____________P
4- 45 58 45 43 00 00 00 00 43 4D 44 00 00 04 0B 04 EXEC____CMD______
5- 0B 00 01 00 00 01 04 50 4E 05 57 44 49 53 4B 00 _______PN_WDISK__
6- 43 4D 44 00 00 04 0C 05 03 00 07 00 00 01 04 50 CMD_____________P
7- 53 41 56 45 00 00 00 00 43 4D 44 00 00 05 04 05 SAV_____CMD______
8- 05 00 02 00 00 01 04 50 54 54 59 53 45 54 00 00 _______PTTYSET___
9- 43 4D 44 00 00 05 06 05 07 00 02 00 00 01 04 50 CMD_____________P
A- 4F 00 00 00 00 00 00 00 43 4D 44 00 00 05 08 05 O_______CMD______
B- 09 00 02 00 00 01 04 50 50 55 43 43 4C 49 4E 4B _______PPUCCLINK
C- 43 4D 44 00 00 05 0A 06 08 00 0E 00 00 01 04 50 CMD_____________P
D- 4A 55 4D 50 00 00 00 00 43 4D 44 00 00 06 09 06 JUMP____CMD______
E- 09 00 01 00 00 01 04 50 44 41 54 45 00 00 00 00 _______PDATE_____
F- 43 4D 44 00 00 06 0A 06 0B 00 02 00 00 01 04 50 CMD_____________P

                                          <It's in this sector.
                                           NEWDISK has the "E" damaged.
                                           Byte 59 should be $45, not $05.>

COMMAND: M,59                             <Modify starting at byte 59.>
59 05 45                                  <Enter the correct value.>
5A 57                                     <Carriage return typed to exit.>
COMMAND: W                                <Re-write the directory sector.>
COMMAND: D                                <Read and dump it to make sure
                                           it's correct.>

DISK ADDRESS: 0006
   -0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -A -B -C -D -E -F
0- 00 07 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
1- 41 50 50 45 4E 44 00 00 43 4D 44 00 00 04 07 04 APPEND__CMD______
2- 09 00 03 00 00 01 04 50 42 55 49 4C 44 00 00 00 _______PBUILD____
3- 43 4D 44 00 00 04 0A 04 0A 00 01 00 00 01 04 50 CMD_____________P
4- 45 58 45 43 00 00 00 00 43 4D 44 00 00 04 0B 04 EXEC____CMD______
5- 0B 00 01 00 00 01 04 50 4E 45 57 44 49 53 4B 00 _______PNEWDISK__
6- 43 4D 44 00 00 04 0C 05 03 00 07 00 00 01 04 50 CMD_____________P
7- 53 41 56 45 00 00 00 00 43 4D 44 00 00 05 04 05 SAV_____CMD______
8- 05 00 02 00 00 01 04 50 54 54 59 53 45 54 00 00 _______PTTYSET___
9- 43 4D 44 00 00 05 06 05 07 00 02 00 00 01 04 50 CMD_____________P
A- 4F 00 00 00 00 00 00 00 43 4D 44 00 00 05 08 05 O_______CMD______
B- 09 00 02 00 00 01 04 50 50 55 43 43 4C 49 4E 4B ________PPUCCLINK
C- 43 4D 44 00 00 05 0A 06 08 00 0E 00 00 01 04 50 CMD_____________P
D- 4A 55 4D 50 00 00 00 00 43 4D 44 00 00 06 09 06 JUMP____CMD______
E- 09 00 01 00 00 01 04 50 44 41 54 45 00 00 00 00 _______PDATE_____
F- 43 4D 44 00 00 06 0A 06 0B 00 02 00 00 01 04 50 CMD_____________P

                                          <Data is correct.>
COMMAND:  S                               <Return to FLEX>
```

As an example of the use of the "T" command, let us assume that it is desired to copy the data from sector 0104 into sector 050F. Let us further assume that only the data must be copied, not the link or record number. Here is how the session might go.

```
COMMAND: R,104                  <Read in sector 0104>
COMMAND: T,*D,4000              <Move the data to a safe area>
COMMAND: R,50F                  <Read in the sector to be written>
COMMAND: T,4000,*D              <Move the data into the buffer>
COMMAND: W                      <Update the sector on the disk>
COMMAND: S                      <Return to FLEX>
```

Program Name: FILETEST
Program Type: DISKETTE DIAGNOSTIC

PURPOSE:

FILETEST tests all or part of a diskette for errors by reading the data files on the diskette. All files, or a specified list of files, may be tested. Optionally, the boot sector and system sectors, the directory chain, and the free chain may be tested. If desired, a list of the sector addresses of those sectors in a file may be displayed. If an error is detected, the name of the file and the nature of the error are displayed.

Calling Sequence:

FILETEST,drive-and-options, file-list

where:

"drive and options" are the number of the drive to be tested and the test options. Either may be specified first, and either or both may be omitted. If no drive is specified, the work drive is used if it is not "all". If the work drive is "all", a drive number must be specified. The options are a string, starting with a "+", composed of one or more of the following letters:

A - Test all files on diskette
D - Test directory chain
F - Test free chain
M - Print map (sector list) for each selected file
S - Test Boot and System sectors

"file list" is a list of FLEX files to be tested. Each file in the list will be tested. If no file list is specified, only those portions of the diskette specified through options will be tested.

METHOD:

When called, FILETEST first checks to see if the boot and system sectors, the directory, or the free chain is to be tested. If so, they are checked before any files are tested. If an error is detected when testing these special areas, the following pseudo-file names are used in the error message:

$BOOT.SYS - Boot Sector
$SYSINFO.SYS - System Information Sector
$DIRECTY.SYS - Directory
$FREECHN.SYS - Free Chain

Each file chain is tested by following the sector links in that chain.

Once an error is found in a file, the testing of that file stops, and testing of the next file begins. It is assumed that the file chains on the diskette are structurally intact; that is, none of the sectors have bad links.

If a file chain can be read without error, the file size and last sector number are compared to that which is stored in the directory. An error message is issued if there is a discrepancy.

MESSAGES:

TESTING COMPLETED
: Testing of the requested files is finished.

LAST SECTOR ERROR, EXPECTED tt/ss, ACTUAL tt/ss
: The number of the last sector of the file, as recorded in the directory, does not correspond to the number of the last sector read by FILETEST. The value after EXPECTED is the value from the directory; that after ACTUAL, the value from the file itself.

SECTOR COUNT ERROR, EXPECTED nnnn, ACTUAL nnnn
: The number of sectors in the file, as recorded in the directory, does not correspond to the count accumulated while reading the file. The value after EXPECTED is the value from the directory; that after ACTUAL, the count accumulated while reading the file.

ERROR READING DIRECTORY
: A disk error was encountered while trying to open a file for testing.

File name NOT FOUND
: The indicated file, which was specified in the file list, could not be found in the diskette directory.

File name CANNOT LOCATE TRACK/SECTOR tt/ss
: When reading the indicated file, the disk controller could not locate a sector. This is usually an indication that the address field of the sector is damaged. Normally, there is nothing that can be done to recover the data from a sector that has this error.

File name READ ERROR TRACK/SECTOR tt/ss
: A CRC error (checksum error) 'was detected by the disk controller when reading the indicated file. With this kind of error, the data might be recoverable.

UNKNOWN OPTION IGNORED - x
: The option "x" is not a valid option. The valid options are: A, D, F, M, and S.

ERROR IN DRIVE NUMBER
: An illegal drive number was specified to the diagnostic as a parameter.

DRIVE MUST BE SPECIFIED
The diagnostic was called without an argument and the default work drive was set to "all". The diagnostic will check only one drive, so in this case, a drive must be specified.

DRIVE NOT READY
The drive is not ready, the diagnostic is aborted.

File name NULL FILE
The "M" option was selected (print list of sector numbers), but the file size was zero. Such a file may be created by resetting the machine with a file open for writing, then re-booting the system.

REMARKS:

FILETEST cannot detect the error in which a file loops back on itself. If FILETEST is reading such a file, it will loop forever. VALIDATE may be used to detect such files.

EXAMPLES

1) Test the file DATA.TXT on the working drive.

```
FILETEST DATA.TXT
```

Note that no drive number is necessary if the diskette is in the working drive (so long as the work drive is not set to "all").

2) Test all of the files on the diskette in drive 0.

```
FILETEST +A 0   or
FILETEST 0 +A
```

Note here that the options and drive number are interchangeable.

3) Print out a list of the sectors comprising the free chain.

```
FILETEST +MF
```

4) Test the directory and the file DATA.DAT on the tiorking drive.

```
FILETEST +D DATA.DAT
```

Program Name: FLAW
Program Type: DISKETTE UTILITY

PURPOSE:

FLAW removes sectors from the free chain of a FLEX diskette either because they contain errors, or because they were specified as parameters to the program. The removed sectors are no longer available for use. This utility is best used for removing sectors that have suddenly gone bad, or for removing sectors that are intermittent or pattern sensitive. These types of sectors normally may be initialized without error, but give errors when they are used in a file.

Calling Sequence:

FLAW,drive-number,sector-list

where:

"drive number" is the drive, containing the diskette to be processed. The diskette must already be mounted. If no drive number is specified, the work drive is used if it has not been set to "all". If the work drive is "all", a drive must be specified.

"sector list" is an optional list of sector addresses, each in the form "ttss" ("tt" is the track number, "ss" is the sector within that track). If the sector list is omitted, only those sectors which contain errors will be removed. Any sector that has an error will be removed, even if it is not in the list.

METHOD:

FLAW starts reading the free chain on the diskette. If a sector is encountered which is specified in the list, it is removed from the chain. If a sector has an error, an informative message is issued, and FLAW attempts to remove it from the chain. If the sector with the error has a bad link, it cannot be removed, and FLAW terminates immediately.

MESSAGES:

ILLEGAL DRIVE NUMBER
An illegal drive number was specified to the diagnostic as a parameter.

DRIVE NUMBER MUST BE SPECIFIED
The diagnostic was called without a drive number and the default work drive was set to "all". The diagnostic will check only one drive, so in this case, a drive must be specified.

"FLAW" FINISHED
The removal of the specified sectors was successfully accomplished.

SYSTEM SECTOR NOT UPDATED
FREE CHAIN DAMAGED
A disk error was encountered when trying to update the system information record with the new configuration of the free chain. The data on the diskette should be copied to another diskette, and the old diskette initialized.

ERROR WHILE READING ttss
An error was detected reading the sector at sector address "ttss". FLAW will attempt to remove the sector from the free chain.

BAD LINK IN ttss - FATAL ERROR
The sector at sector address "ttss" contains a bad link. It is not possible to remove it from the free chain, nor is it possible for FLAW to continue. FLAW terminates immediately.

FREE CHAIN ENDS PREMATURELY - FATAL ERROR
The end of the free chain was encountered unexpectedly. The free chain is probably shorter than indicated in the system information sector. The diskette should be considered structurally damaged, and the data copied to a good diskette.

END OF FREE CHAIN NOT FOUND
The free chain either is longer than indicated in the system information sector, or loops back on itself. The diskette should be considered structurally damaged, and the data copied to a good diskette.

SYSTEM SECTOR CANNOT BE READ
A disk error was encountered when trying to read the system information sector. FLAW terminates immediately.

ttss NOT FLAWED
The sector indicated by "ttss", which was specified as a parameter, was not found in the free chain.

CANNOT UPDATE ttss - FATAL ERROR
When trying to remove a sector from the free chain, the sector pointing to the one being removed could not be read to be updated. This sector had been read previously without error. FLAW terminates immediately. FLAW may be re-run to remove this sector.

ERROR DURING UPDATE OF ttss - FATAL ERROR
When trying to remove a sector from the free chain, the sector pointing to the one being removed yielded a write error when it was considered structurally damaged, and the data copied to a good diskette.

REMARKS:

FLAW assumes that the system information sector is intact. It is from this sector that the information on the free chain and diskette configuration is obtained. If this sector cannot be read, FLAW issues a message and terminates.

The free chain should not contain sectors which have a damaged address field. These sectors would result in a "sector not found" error from the TEST or FILETEST diagnostic. FLAW does not distinguish among the various types of errors and will try to remove a "not found" sector from the free chain, resulting in structural damage to the free chain.

FLAW updates the system information sector each time a sector is removed. If FLAW should terminate before processing the entire free chain, the free chain is probably intact. However, under these circumstances, it would be wise to check the diskette with VALIDATE.

After FLAW has run, it is good practice to run VALIDATE. If a bad sector has a damaged link field, but the link is still within the range of legal values, then FLAW may cause structural damage to the files or free chain.

EXAMPLES

1) Remove any sectors in the free chain of the diskette in the work drive that have errors.

```
FLAW
```

2) Remove any sectors in the free chain of the diskette in drive 1 that have errors.

```
FLAW 1
```

2) Remove any sectors in the free chain of the diskette in the work drive that have errors. Also remove sectors 0103, 050F, and 1B05.

```
FLAW,,0103,050F,1B05
```

Note that two commas were necessary after FLAW because a parameter must be reserved for the drive number. Since no drive number was actually specified, the work drive is used. (In this case, the work drive must not be set to "all").

Program Name: RAWCOPY
Program Type: DISKETTE UTILITY

PURPOSE:

RAWCOPY copies a file, ignoring checksum errors (CRC errors) whenever possible. It is intended to be used in an attempt to retrieve most of the data in a file that has a bad sector in it. Because of the checksum error, the data in that sector will be damaged; however, it is assumed that once a readable copy is available, an editor or the EXAMINE utility can be used to correct the damage.

Calling Sequence:

RAWCOPY old-file,new-file

where:

"old-file" is the file specification of the file that contains the bad sector. The default extension is TXT".

"new-file" is the specification of the file that is to be written. The default extension is that of the old file.

METHOD:

RAWCOPY performs a simple file copy function, ignoring checksum errors in the file being copied, if possible. The file is copied one sector at a time. The link in each sector is validated against the legal maxima for track and sector. These values are read from the system information sector of the diskette containing the bad file. If the system information sector cannot be read, the user will be prompted for information sufficient to determine the size and configuration of the diskette. If, while reading the file, a bad link is detected, or a "sector not found" or "drive not ready" error is detected, the copy is aborted.

MESSAGES:

COPY COMPLETED
The copy operation has terminated normally.

COPY ABORTED
The copy operation could not be completed because either a bad link was detected in a sector or a "not found" or "not ready" error was detected.

SYSTEM INFO SECTOR INVALID
The diagnostic could not read the system information sector on the diskette, or the information read concerning maximum track and sector did not appear correct. This message will be followed by prompts for disk configuration information. See "The Disk Utilities in This Package: System Dependencies" for details.

ADDRESS: ttss, DRIVE NOT READY
A "not ready" response was received from the disk controller when the sector at disk address "ttss" was being read.

ADDRESS: ttss, SECTOR NOT FOUND
The sector specified by disk address "ttss" could not be located by the disk controller. This normally indicates damage to the address portion of the sector.

ADDRESS: ttss, CRC ERROR
A checksum error was detected by the disk controller while reading the sector at disk address "ttss".

BAD LINK ENCOUNTERED
A bad link to the next sector was detected while reading the file. The copy is aborted.

Any other messages are produced by the FLEX operating system.

Program Name: REBUILD
Program Type: DISKETTE UTILITY

PURPOSE:

REBUILD attempts to find files on a crashed diskette whose directory has been destroyed. Those files that are located are copied to another drive.

Calling Sequence:

REBUILD source-drive,destination-drive

where:

"source drive" is the number of the drive containing the crashed diskette.

"destination drive" is the number of the drive containing a diskette which will receive copies of those files that can be located.

REBUILD will pause before starting the recovery so that the appropriate diskettes can be inserted in the drives.

METHOD:

REBUILD starts at track 1, sector 1 and searches the diskette for a sector which has a record number of 0001. When one is found, it is assumed to be the start of a file. The chain, starting at that sector, is read to determine if it really is a file. If the chain consists of a series of sectors with record numbers that are in order, then a file has been found. If the record numbers are not correct, it is assumed that this file had been deleted and is now in the free chain; it is not recovered. Once the file has been found, it is copied to the other diskette. A name of the form FILEnnnn.SCR is assigned to the copy, where "nnnn" is an increasing number. The first file found is given the name FILE0001.SCR; the second, FILE0002.SCR, etc. After the file has been copied, or a chain was found not to be a legitimate file, then the scan continues from where it found the first sector of the chain. After the files have been recovered, it is up to the user to list or dump them to determine what the files are, and to rename them appropriately.

MESSAGES:

RECOVERY COMPLETED
REBUILD has found all of the files on the diskette.

COPY ABORTED
The current file being copied has a disk error in it. The copying of the file is terminated and the search for another file resumes.

SYSTEM INFO SECTOR INVALID
The diagnostic could not read the system information sector on the diskette, or the information read concerning maximum track and sector did not appear correct. This message will be followed by prompts for disk configuration information. See "The Disk Utilities in This Package: System Dependencies" for details.

DRIVE NUMBER ERROR
An illegal drive number was specified to the diagnostic as a parameter, or a drive number was missing.

DRIVES ARE THE SAME
The "source drive" and "destination drive" may not be the same.

ADDRESS: ttss, DRIVE NOT READY
A "not ready" response was received from the disk controller when the sector at disk address "ttss" was being read.

ADDRESS: ttss, SECTOR NOT FOUND
The sector specified by disk address "ttss" could not be located by the disk controller. This normally indicates damage to the address portion of the sector.

ADDRESS: ttss, CRC ERROR
A checksum error was detected by the disk controller while reading the sector at disk address "ttss".

FILEnnnn.SCR ttss nn SECTOR(S)
While the file is being copied, its name, FILEnnnn.SCR (where nnnn is a number) is displayed. Also displayed is the starting track and sector ("ttss") of the file on the crashed diskette. When the copy is completed, the number of sectors copied is also displayed.

INSERT DISKS, HIT ANY KEY
When this message appears, insert the appropriate diskettes in their drives and type any key. The recovery process will then begin.

REMARKS:

If a file was a random file on the crashed diskette, the file sector map will not be recovered by REBUILD. After the file has been copied, it must be recopied from the good diskette with the COPYR utility in order to rebuild the file sector map.

The file most recently deleted, if still intact in the free chain, will also be recovered as a separate file.

Program Name: RECOVER
Program Type: DISKETTE UTILITY

PURPOSE:

RECOVER copies files from a crashed diskette to another diskette. Files to be copied are specified by their starting track and sector.

Calling Sequence:

RECOVER source-drive,destination-drive

where:

"source drive" is the number of the drive containing the crashed diskette.

"destination drive" is the number of the drive containing a diskette which will receive copies of the specified files.

RECOVER will pause before starting the recovery so that the appropriate diskettes can be inserted in the drives.

METHOD:

After the diskettes have been mounted, RECOVER prompts for the disk address of the file to be copied. The disk address should be entered in the form: ttss, where "tt" is the the track number in hexadecimal, and "ss" is the sector number in hexadecimal. After the disk address has been entered, RECOVER prompts for the name to be given to the copy of the file. The default extension is ".TXT". Once this information has been entered, the file is copied. After the copy, a prompt for another disk address is issued. To exit from RECOVER, enter a carriage return in answer to the prompt for a disk address.

The copy is performed by following the links in the file chain. No validation of sector links or record numbers is performed.

MESSAGES:

COPY ABORTED
A read error was encountered while copying the file. The copy of the file is abandoned.

DRIVE NUMBER ERROR
An illegal drive number was specified to the diagnostic as a parameter, or a drive number was missing.

ADDRESS: ttss, DRIVE NOT READY
A "not ready" response was received from the disk controller when the sector at disk address "ttss" was being read.

ADDRESS: ttss, SECTOR NOT FOUND
The sector specified by disk address "ttss" could not be located by the disk controller. This normally indicates damage to the address portion of the sector.

ADDRESS: ttss, CRC ERROR
A checksum error was detected by the disk controller while reading the sector at disk address "ttss".

INSERT DISKS, HIT ANY KEY
When this message appears, insert the appropriate diskettes in their drives and type any key. The recovery process will then begin.

DISK ADDRESS:
This is the prompt for the disk address of the start of the file to be recovered.

FILE NAME:
This is a prompt for the name to be assigned to the copy of the file.

ERROR IN ADDRESS
The disk address typed was not a valid hexadecimal number.

FILE NAME ERROR
The file name typed was not a valid FLEX file name.

REMARKS:

RECOVER does not attempt to detect if the file being copied is a random file. If the file you are going to recover is a random file, the file sector map associated with the file will be copied along with the data, but the sector map will no longer be correct. To correct this, the old sector map has to be removed, and a new one constructed. Since this involves the use of more than one utility, it will be covered in one of the cases in the section "Case Studies".

EXAMPLE

Recover the files starting at addresses 0306 and 070A. The crashed disk is in drive 1, and the copies are to be put on the diskette in drive 0. Give the first file the name DATA.DAT; the second, TRIAL.BAS.

```
+++RECOVER 1 0                          <Recover from 1 to 0>
INSERT DISKETTES, HIT ANY KEY
DISK ADDRESS: 0306                      <Enter address for first file>
FILE NAME: DATA.DAT                     <Enter name for first file.
```

```
                                          The file is recovered.>
DISK ADDRESS: 070A                       <Address for second file>
FILE NAME: TRIAL.BAS                     <Enter name for second file.
                                          The file is recovered.>
DISK ADDRESS:                            <Carriage return typed>
RECOVERY COMPLETED
```

Program Name: TEST
Program Type: DISKETTE DIAGNOSTIC

PURPOSE:

TEST reads every sector on a diskette, reporting those that have errors.

Calling Sequence:

TEST drive-number

where:

"drive number" is the drive containing the diskette to be tested. The diskette must already be mounted. If no drive number is specified, the work drive is used if it has not been set to "all". If the work drive is "all", a drive must be specified.

METHOD:

TEST first reads the system information sector on the diskette to determine the number of tracks, number of sectors per track number of sides, and density. If the system information sector cannot be read, the user is prompted for the information. TEST then starts at the track 0, sector 1, and reads each sector on the diskette. The address of each sector that has an error is reported, along with the type of error encountered.

MESSAGES:

SYSTEM INFO SECTOR INVALID
The diagnostic could not read the system information sector on the diskette, or the information read concerning maximum track and sector did not appear correct. This message will be followed by prompts for disk configuration information. See "The Disk Utilities in This Package: System Dependencies" for details.

ILLEGAL DRIVE NUMBER
An illegal drive number was specified to the diagnostic as a parameter.

DRIVE MUST BE SPECIFIED
The diagnostic was called without an argument and the default work drive was set to "all". The diagnostic will check only one drive, so in this case, a drive must be specified.

ADDRESS: ttss, DRIVE NOT READY

A "not ready" response was received from the disk controller when the sector at disk address "ttss" was being read. The test terminates immediately.

ADDRESS: ttss, SECTOR NOT FOUND

The sector specified by disk address "ttss" could not be located by the disk controller. This normally indicates damage to the address portion of the sector.

ADDRESS: ttss, CRC ERROR

A checksum error was detected by the disk controller while reading the sector at disk address "ttss".

REMARKS:

Some disk controllers used for 5" diskette drives do not have the capability of detecting that a drive is not ready. If the drive is not ready, the test will hang until the drive is made ready.

Program Name: UNDELETE
Program Type: DISKETTE UTILITY

PURPOSE:

UNDELETE attempts to remove deleted files from the free chain, restoring them in the directory with a user specified name.

Calling Sequence:

UNDELETE drive-number

where:

"drive number" is the drive containing the diskette which must already be mounted. If no drive number is specified, the work drive is used if it has not been set to "all". If the work drive is "all", a drive must be specified.

METHOD:

UNDELETE starts by searching the free chain, building a map of those sector chains that appear to constitute files. After the scan is finished, UNDELETE is ready to process the files that it has found. The files that it has found in the free chain are numbered, with the "youngest" file (the most recently deleted) being number 1. By using commands, the user may examine and optionally recover any file from the free chain. The youngest file in the free chain is made the "current" file, and information about that file is displayed The information displayed includes the size,of the file, the starting disk address and whether it is a sequential file or a random file. UNDELETE then waits for a command. The possible commands are:

D - Dump the current file is hexadecimal and ASCII.
N - Proceed to the next older file.
P - Go back to the previous (next younger) file.
R - Recover the current file. A prompt for the file name will follow.
S - Return to FLEX.

In addition to the above commands, the number of a file may be typed. That file is then made the "current" file, and the information about that file is displayed.

Commands:

UNDELETE indiactes that it is ready for a command by issuing the prompt:

ACTION (D/N/P/R/S/#)?

The letters in parentheses are the legal commands, with "#" meaning that a file number may be entered.

D - Dump the Current File
The content of the current file is dumped in hexadecimal and ASCII. The escape key may be used to temporarily stop the dump. Typing an escape followed by a return will cause the ACTION prompt to be re-issued.

N - Proceed to the next Older File.
The next older file (closer to the beginning of the free chain) is made the current file and information about that file is displayed.

P - Go back to the Previous File.
The next younger file (closer to the end of the free chain) is made the current file and information about that file is displayed.

R - Recover the Current File.
The current file is to be recovered. The user is prompted for the name to be assigned to the recovered file. At this point, a name must be typed since the file has already been removed from the free chain. If an illegal name is given, the request for a name will be re-issued.

S - Return to FLEX

If a carriage return is entered, the information about the current file is re-displayed.

MESSAGES:

ACTION (D/N/P/R/S/#)?
The prompt for the next command.

DRIVE MUST BE SPECIFIED
No parameter was specified when UNDELETE was called, and the working drive was set to "ALL". If the working drive is "ALL", a drive number must be specified as a parameter when calling UNDELETE.

ILLEGAL DRIVE SPECIFIED
An invalid drive number was specified as the parameter to EXAMINE.

FILE ALREADY EXISTS
The file name typed already exists in the diskette directory. A file name is requested again.

nnnn FILES FOUND
This message is issued after the scan of the free chain is completed. It indicates the number of files found.

FILE NUMBER OUT OF RANGE
A file number was entered in response to the ACTION prompt that was larger than the number of files found in the free chain. Entering a file number of zero will also result in this message.

UNRECOGNIZED COMMAND
The command that was typed could not be recognized.

FREE CHAIN IS EMPTY
There are no files in the free chain.

FILE NAME?
This is a request for the name to be assigned to the recovered file. The default extension in ".BIN".

REMARKS:

The scanning of the free chain for files may take a long time, especially on double sided or double density diskettes.

Only complete files may be recovered. If a part of a deleted file has already been re-used by FLEX, that file can not be recovered.

EXAMPLE

Assume that we want to recover a file that has been deleted recently. We know that it was about 20 sectors long, and that it was a sequential file containing the source of a program called LOAD. If the diskette that has the file is in drive 1, then the recovery session night proceed as follows:

```
+++UNDELETE 1                          <Call UNDELETE>
6 FILES FOUND
FILE 1 6 SECTOR(S), ADDRESS: 0206 TYPE: SEQUENTIAL
ACTION (D/N/P/R/S/#)? N                <Go to the next file>
FILE 2  19 SECTOR(S), ADDRESS: 0F04 TYPE: RANDOM
ACTION (D/N/P/R/S/#)? N                 <Go to the next file, this one is
                                         about the right size, but it's
                                         a random file, not sequential>
FILE 3  1 SECTOR(S), ADDRESS 0102 TYPE: SEQUENTIAL
ACTION (D/N/P/R/S/#)? N                <Go to the next file>
FILE 4  22 SECTORS(S), ADDRESS 300F TYPE: SEQUENTIAL
ACTION (D/N/P/R/S/#)? D                <This might be it. Dump it.>

30 OF
30 10 00 01 20 4E 41 4D 20 4C 4F 41 44 20 2D 20 0___NAM LOAD -
4C 4F 41 44 20 22 53 31 22 20 46 49 4C 45 20 46 LOAD "S1" FILE F
52 4F 4D 20 43 41 53 53 45 54 54 45 2E 00 20 4F ROM CASSETTE._ O
50 54 20 50 41 47 0D 20 50 41 47 0D 2A 2A 2A 09 PT PAG_ PAG_***
03 4C 4F 41 44 20 2D 20 4C 4F 41 44 20 22 53 31 _LOAD - LOAD "S1
22 20 46 49 4C 45 20 46 52 4F 4D 20 43 41 53 53 " FILE FROM CASS
45 54 54 45 2E 0D 20 53 50 43 20 34 0D 2A 2A 09 ETTE._ SPC 4_**_
```

```
03 53 59 4D 42 4F 4C 20 44 45 46 49 4E 49 54 49  SYMBOL DEFINITI
4F 4E 53 2E 0D 20 53 50 43 20 32 0D 47 45 54 43 ONS.  SPC 2 GETC
48 52 20 45 51 55 20 24 43 44 31 35 20 47 45 54 HR EQU $CD15 GET
20 43 48 41 52 41 43 54 45 52 0D 50 43 52 4C 46  CHARACTER PCRLF
20 45 51 55 20 24 43 44 32 34 20 50 52 49 4E 54  EQU $CD24 PRINT
20 43 52 2F 4C 46 0D 50 53 54 52 4E 47 20 45 51  CR/LF PSTRNG EQ
55 20 24 43 44 31 45 20 50 52 49 4E 54 20 53 54 U $CD1E PRINT ST
52 49 4E 47 0D 50 55 54 43 48 52 20 45 51 55 20 RING PUTCHR EQU
24 43 44 31 38 20 4F 55 54 50 55 54 20 43 48 41 $CD18 OUTPUT CHA

30 10
30 11 00 02 52 41 43 54 45 52 0D 57 41 52 4D 53 0   RACTER WARMS
20 45 51 55 20 24 43 44 30 33 20 52 45 54 55 52  EQU $CD03 RETUR
4E 20 54 4F 20 53 59 53 54 45 4D 0D 20 53 50 43 N TO SYSTEM  SPC
```

```
                                        <Printing stopped by typing
                                         "escape" followed by "return">
ACTION (D/N/P/R/S/#)? R                 <This is the file, recover it>
FILE NAME? LOAD.TXT                     <Name is LOAD.TXT>
ACTION (D/N/P/R/S/#)? S                 <Return to FLEX>
```

Program Name: VALIDATE
Program Type: DISKETTE DIAGNOSTIC

PURPOSE:

VALIDATE checks a FLEX diskette for structural errors caused by hardware or software malfunction. The following items are checked:

a) that the sector links in each file are legal track and sector values,
b) that the record numbers in the sectors of a file are correct,
c) that the sectors in a random file correspond to those specified in the file sector map,
d) that the file size and ending disk address correspond to the values in the directory,
e) that the free chain corresponds to its description in the system information record,
f) that the directory does not end prematurely,
g) that files do not intersect, and
h) that there are no orphaned sectors (viz. those that are not in a file nor in the free chain or directory).

Calling Sequence:

VALIDATE,drive-number

where:

"drive number" is the drive containing the diskette to be validated. The diskette must already be mounted. If no drive number is specified, the work drive is used if it has not been set to "all". If the work drive is "all", a drive must be specified.

METHOD:

VALIDATE reads every file chain on the diskette, including the directory and the free chain. A record of each sector in the chain is made in a table in memory. As each chain is scanned, the table is checked to determine if the sector currently being read was part of another chain. If so, this is an error since the two chains intersect. As each file is being read, the links in each sector are checked against the values permitted for the size of diskette being tested. Any track and sector values which are out of range are reported as an error. In addition, the record number in each sector is checked. Record numbers that are out of sequence are also reported as errors. If the directory indicates that the file being checked is a random file, the file sector map is checked for valid structure. Each sector in the file must also be in the sector map. Any discrepancies are reported as errors. After all of the file chains have been examined, the table in memory is examined to determine if any sectors have not been encountered. If some have been

missed, a count of them is printed.

MESSAGES:

FILE: file name, LAST SECTOR ERROR
The last sector of the indicated file did not correspond to that specified in the directory.

FILE: file name, FILE SIZE ERROR
The size of the indicated file did not correspond to the value specified in the directory.

SECTORS NOT FOUND: nn
"nn" is a count of the number of sectors which were not found during the validation process.

SYSTEM INFO SECTOR INVALID
The diagnostic could not read the system information sector on the diskette.

ADDRESS ttss, PREMATURE END OF DIRECTORY
At disk address "ttss" in the directory, a zero entry was found indicating the end of the directory. However, additional directory entries were found beyond that point.

FILE: file name, ADDRESS: ttss, ILLEGAL SECTOR MAP
The directory entry for the file indicated that the file was a random file. However, the file sector map at disk address "ttss" did not have a zero record number, as required.

FILE CONFLICT: file name/file name
The specified files intersect. One of the files will probably be named in another error message.

ILLEGAL DRIVE NUMBER
An illegal drive number was specified to the diagnostic as a parameter.

DRIVE MUST BE SPECIFIED
The diagnostic was called without an argument and the default work drive was set to "all". The diagnostic will check only one drive, so in this case, a drive must be specified.

FILE: file name, ADDRESS ttss, BAD LINK
The sector at disk address "ttss" in.the specified file contains a forward link that is outside of the permissible value for a track and sector for the type of diskette being tested.

ADDRESS: ttss, DRIVE NOT READY
A "not ready" response was received from the disk controller when the sector at disk address "ttss" was being read.

ADDRESS: ttss, SECTOR NOT FOUND
The sector specified by disk address "ttss" could not be located by the disk controller. This normally indicates damage to the address portion of the sector.

ADDRESS: ttss, CRC ERROR
A checksum error was detected by the disk controller while reading the sector at disk address "ttss".

FILE: file name, NULL FILE
The specified file contains no data.

FILE file name, ADDRESS ttss, RECORD NUMBER ERROR
The sector at disk address "ttss" in the specified file did not contain the expected record number. Record numbers should increase by one along the length of the file.

FILE file name, ADDRESS ttss, SECTOR MAP ERROR
The sector at disk address "ttss" in the indicated random file contains a link to a sector which is not in the file sector map.

VALIDATION COMPLETED
The diagnostic is finished.

VALIDATION ABORTED
The diagnostic detected an error of such a magnitude that it could not complete its task. Such errors include: checksum error, a sector could not be found by the disk controller, the drive not being ready, and the system information sector being damaged.

REMARKS:

VALIDATE assumes that the diskette does not contain files with checksum errors or sectors that cannot be located by the disk controller. If any such sectors are found, a message is issued and the diagnostic is aborted. The routines TEST and FILETEST can be used to determine if any such sectors exist. If the system information sector cannot be read, or contains anomalous values, the diagnostic is also aborted.

Bad sectors that were removed by NEWDISK or FLAW will be included in the count of sectors that were not found since they were not in the free chain or a file. The number of bad sectors should be subtracted from. this count to determine how many "orphaned" sectors there are.

This diagnostic takes 2 minutes to check an 8" single-sided, single-density diskette. Double-sided and double-density diskettes take proportionately longer.

Some controllers for 5 1/4" diskettes will hang if the the drive is not ready or the sector cannot be located by the controller. Owners of such hardware should be aware that this might be the cause of the diagnostic apparently hanging up.

VALIDATE will only function correctly on the diskette configurations listed in the "System Dependencies" paragraph in the section "The Disk Utilities in this Package". It will not work on a hard disk.

CASE STUDIES

CASE STUDIES

The following examples are intended to demonstrate how the disk utilities in this package can be used to ident@ify and sometimes correct problems. It should be stressed that these programs cannot correct all problems. There are situations in which data has been destroyed and cannot be salvaged. Proper identification of the problem is important in determining if the data can be saved. Such identification must be based on interpretation of the messages issued by the diagnostics. This, as well as some tricks, are stressed in the cases considered.

When attempting to salvage damaged data, it is important to take into consideration the amount of work involved in the recovery process. It may be faster to reconstruct the data from backups than it would be to salvage the damaged data and repair it. After you have recovered damaged data a few times you will be able to judge which option is best.

CASE I: A Simple Read Error

Assume that while assembling a program, you get the message DISK FILE READ ERROR while reading the file LOAD.TXT from the disk in drive 1. The disk containing the utilities is in drive 0. The problem is to recover as much data as possible from the file.

The first step is to try to read the file on another disk drive. The mechanical differences among drives are sometimes enough to enable one to read an intermittent bad spot that another drive cannot read. The FILETEST utility can be used to read the file. If it does not get an error, the file can then be copied to another disk. If it does get an error, then FILETEST will tell you the disk address that has the error. Switching the system and work disks, we try FILETEST.

```
+++1.FILETEST 0 LOAD.TXT                <Remember, we switched disks so we
                                         specify the drive numbers.>
0.LOAD.TXT READ ERROR TRACK/SECTOR 04/07
TESTING COMPLETED
```

The file cannot be read on another drive, so we have to try to recover the data. Remember the track and sector containing the error, we'll come back to it later. Putting the disks back in the original drives, copy the file using RAWCOPY.

```
+++RAWCOPY LOAD.TXT LOADX.TXT
ADDRESS: 0407, CRC ERROR
COPY COMPLETED
```

The file LOADX.TXT now contains a readable copy. However, there is probably some damage to the data in that file because of the error detected while reading the old file. If the file is short it would be easiest to bring up an editor and look for the damaged data. On the other hand, if the file is long- the damage might be hard to find. The EXAMINE utility can be used to read the bad sector, which will give us some idea of where the damage is located. By looking at the data in the bad sector, we may be able to see what the damage is like and where in the program the damaged code is located. From this information, we can go to the same spot in the good copy of the file to see what needs to be corrected.

```
+++EXAMINE                                  <Default to the work drive>
COMMAND: D 0407                             <Dump the bad Sector>
ADDRESS: 0407, CRC ERROR

DISK ADDRESS: 0407
   -0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -A -B -C -D -E -F
0- 04 08 00 0A 54 0D 20 42 56 53 20 42 59 54 45 31 ____T_ BVS BYTE1
1- 20 49 46 20 45 52 52 4F 52 0D 20 50 53 40 53 20  IF ERROR_ PS@S
2- 42 20 41 53 53 45 4D 42 4C 45 20 42 59 54 45 0D B ASSEMBLE BYTE_
3- 20 41 44 44 41 20 30 2C 53 2B 0D 42 59 54 45 31  ADDA 0,S+ BYTE1
4- 20 52 54 53 20 52 45 54 55 52 4E 0D 20 53 50 43  RTS RETURN_ SPC
5- 20 34 0D 2A 2A 09 03 44 49 47 20 2D 20 41 53 53  4 ** _DIG - ASS
6- 45 4D 42 4C 45 20 44 49 47 49 54 2E 0D 2A 0D 2A EMBLE DIGIT. * *
7- 09 04 45 58 49 54 09 02 56 53 20 49 46 20 45 52 _EXIT_VS IF ER
8- 52 4F 52 20 44 45 54 45 43 54 45 44 0D 2A 09 0A ROR DETECTED *__
9- 56 53 20 49 46 20 4E 4F 20 45 52 52 4F 52 2C 20 VS IF NO ERROR,
A- 41 4E 44 0D 2A 09 0A 28 41 29 3D 44 49 47 49 54 AND_*__(A)=DIGIT
B- 0D 20 53 50 43 20 32 01 44 49 47 20 42 53 52 20  _SPC 2_ DIG BSR
C- 49 4E 43 48 20 47 45 54 20 43 48 41 52 41 43 54 INCH GET CHARACT
D- 45 52 0D 20 42 56 53 20 44 49 47 31 20 49 46 20 ER BVS DIG1 IF
E- 45 52 52 4F 52 0D 20 53 55 42 41 20 23 27 30 20 ERROR_ SUBA '0
F- 43 48 45 43 4B 20 44 49 47 49 54 0D 20 42 4C 4F CHECK-DIGIT_ BLO

COMMAND:  S
```

At byte $1D, we see that an "H" has been changed to a "@". The rest of the sector looks good, there are no other errors. (it is important to check the whole sector, more than one byte may be damaged.) With this information, we can edit LOADX, look for "PS@S", and change it back to "PSHS".

The problem now is what to do with the bad file. We can simply forget about it, but then the good sectors in the file cannot be reused by other files. If the bad sector is not the first sector, then the good sectors can be reclaimed by deleting the file, and then removing the bad sector from the free chain with the FLAW utility. If the error is in the first sector of the file, it will not be possible to delete it without causing more damage to the disk. By looking at the record number bytes, we see that this is the tenth sector in the file (bytes 2-3 of the sector are 000A). Therefore we can delete it.

```
+++DELETE LOAD.TXT
DELETE "1.LOAD.TXT" ? Y
ARE YOU SURE?  Y
+++FLAW 1                                   <It is not necessary to specify
                                             the bad sector if it has an
                                             error in it.>
ERROR WHILE READING 0407
"FLAW" FINISHED
```

If the bad sector had been the first sector of the file, then the best

course of action would be to copy everything to a good disk and re-initialize the bad one. This may seem like a lot of work just to recover from a read error, but it really does not take a long time.

The most important thing to note in this case is the procedure of first identifying exactly where the problem is located (sector 0407 in the file LOAD.TXT), then recovering the data, and finally, repairing the damage caused by the error.

CASE II: A "Sector Not Found" Error

While reading a disk, a DRIVE NOT READY message appeared, yet the drive was ready. Unfortunately, we don't know which file was being read. The first step is to isolate the problem using FILETEST.

```
+++FILETEST +A                          <Use the "A" option to check
                                         all of the files>
1.ELECTRIC.TXT CANNOT LOCATE 1A/05
TESTING COMPLETED
```

This is a very bad error. It means that the address field in front of the data in the sector cannot be read. In many cases, this type of error produces a "not ready" condition. There is very little that can be done to save the data since the disk controller cannot find the sector.

The address field cannot be repaired, the disk must be re-initialized. The front of the file ELECTRIC.TXT can be salvaged by using RAWCOPY, as in CASE I. The copy will stop when the bad sector is reached. There is a "desperation" technique that can be used to recover the back end of the file, but it is very time-consuming and a lot of work. This is investigated in a later case.

CASE III: Recovering a Random File

A data disk containing a random file has had its directory destroyed. You know from a printed output from the DIR utility (not part of this package), that the file starts at disk address 0204, and is 28 sectors long. The problem is to recover the random file to another disk.

There are two ways to recover the file. The REBUILD utility can recover all of the files on the disk, and the RECOVER utility can recover just the random file. We will look at both ways.

Using the REBUILD utility, we will recover all of the files on the disk.

```
+++REBUILD 0 1                          <Bad disk in 0, good disk in 1>
INSERT DISKS, HIT ANY KEY
1.FILE0001.SCR  0102  10 SECTOR(S)
1.FILE0002.SCR  0206  26 SECTOR(S)
1.FILE0003.SCR  1001  1 SECTOR(S)
1.FILE0004.SCR  1002  16 SECTOR(S)
RECOVERY COMPLETED
```

One of these files is the data from our random file. REBUILD does not recover the file sector map, so the length of the new file will be 2 sectors shorter than the original file. FILE0002.SCR appears to the one. If we are not sure, we could use EXAMINE to dump the file. The disk with the damaged directory can be put aside, it is no longer needed.

The next step is to put a file sector map on the copy of our random file. The COPYR utility is designed to do this.

```
+++COPYR FILE0002,DATA.DAT
```

The file DATA.DAT now contains the random file, in its entirety.

The other method of recovering the random file involves the use of the RECOVER command. RECOVER starts at the disk address that you enter, and copies every sector. We cannot start it at the original first sector (0204), because this is the address of the file sector map. We do not want to copy the file sector map exactly, because it would then not reflect the layout of the file. A file sector map must be built, not copied. The trick is to use EXAMINE to read the file sector map and determine the first sector that contains data.

```
+++EXAMINE                              <Crashed disk in work drive>
COMMAND: D 0204                         <Dump the first sector>

DISK ADDRESS: 0204
   -0 -1 -2 -3 -4 -5 -6 -7 -8 -9 -A -B -C -D -E -F
0- 02 04 00 00 02 06 1A 00 00 00 00 00 00 00 00 00 ________________
1- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
2- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
3- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
4- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
5- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
6- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
7- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
8- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
9- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
A- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
B- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
C- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
D- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
E- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________
F- 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ________________

COMMAND: S
```

By looking at the file sector map, we see that the data itself starts in sector 0206.
This is the value that we will give to RECOVER.

```
+++RECOVER  0 1                         <Bad disk in 0, good disk in 1>
INSERT DISKS, HIT ANY KEY
DISK ADDRESS: 0206                      <Enter the starting address of
                                         the data>
FILE NAME: DATA.SCR                     <Put it on a scratch file>
DISK ADDRESS:                           <Enter carriage return to exit>
```

Now that the data is recovered, the old disk can be put aside. The COPYR utility is used, as in the first part of this case, to build a file sector map on the recovered file.

```
+++COPYR DATA,DATA.DAT
```

The file DATA.DAT now contains the random file in its entirety.

CASE IV: A Structural Problem

While editing the file CPINT, the message ALL AVAILABLE DISK SPACE HAS BEEN USED appeared. There should have been plenty of space on the disk, so there must be a structural problem. Running the CAT command, we get:

```
CATALOG OF DRIVE NUMBER 1
DISK:   #0

 NAME  TYPE SIZE PRT

CPINT  .BAK   95
CPINT  .TXT   47
CPINT  .BIN    4
GTCVT  .CMD    1
GTCVT  .BAK    7
GTCVT  .TXT    7

SECTORS LEFT = 0
```

Comparing the sizes of CPINT.TXT and CPINT.BAK, we see that the file that was being edited is badly truncated. If the .BAK file is still intact, we might be able to salvage something. The first step is to find out the extent of the damage using the VALIDATE command.

```
+++VALIDATE                                <Bad disk is in working drive>
FILE: 1.CPINT.BAK, ADDRESS: 180A, RECORD NUMBER ERROR
FILE: 1.CPINT.BAK, FILE SIZE ERROR
FILE CONFLICT: 1.CPINT.BAK/1.CPINT.TXT
SECTORS NOT FOUND: 1032
VALIDATION COMPLETED
```

The file CPINT.BAK has some serious problems with it. It is obvious that the file has been overwritten or gotten linked into another file. The 1032 sectors that couldn't be located are probably the sectors in the free chain. We can get another look at the damage by printing the file chains using FILETEST.

```
+++FILETEST +M CPINT.TXT CPINT.BAK

1.CPINT.TXT 13/0C 13/0D 13/0E 13/0F 14/01 14/02 14/03 14/04 14/05
   14/06 14/07 14/08 14/09 14/0A 14/0B 14/0C 14/0D 14/0E 14/0F 15/01
   15/02 15/03 15/04 15/05 15/06 15/07 15/08 15/09 15/0A 15/0B 15/0C
   15/0D 15/0E 15/0F 16/01 16/02 16/03 16/04 16/05 16/06 16/07 18/0A
   13/07 13/08 13/09 13/0A 13/0B
```

```
1.CPINT.BAK 1210 9 1210A 12/0B 12/0C 12/0D 12/0E 12/0F 13/01 13/02
   13/03 13/04 13/05 13/06 16/0B 16/0C 16/0D 16/0E 16/0F 17/01 17/02
   17/03 17/04 17/05 17/06 17/07 17/08 17/09 17/0A 17/0B 17/0C 17/0D
   17/0E 17/0F 18/01 18/02 18/03 18/04 18/05 18/06 18/07 18/08 18/09
   18/0A 13/07 13/08 13/09 13/0A 13/0B
1.CPINT.BAK SECTOR COUNT ERROR, EXPECTED 95, ACTUAL 48
```

Our worst fears are justified. Both the .TXT file and the .BAK file have been destroyed. There is no hope of recovering any of the data. Perhaps the "desperation" method, described later on may be useful, but that may be as much work as retyping the entire file.

It is quite likely that the damage to the disk was done some time ago and only manifested itself recently by destroying these two files. Structural problems do not go away, they only get worse. If, at some time in the past, VALIDATE was run on this disk, the problem might have been detected before it destroyed both files. Disks that are heavily used for editing should be checked periodically with VALIDATE, just in case. VALIDATE should definitely be run on a disk if the machine is reset while the disk is being written, or if a power failure occurred while the disk was in the machine. It only takes 5 to 10 minutes to run and it could save a lot of work later on if a structural error is detected early.

CASE V: Rehabilitating a Bad Directory Chain

A new disk could not be formatted by NEWDISK. The message FATAL ERROR - FORMATTING ABORTED always appeared. The problem is to determine if the disk can be salvaged.

The first problem is to determine what kind of problem that we have. The TEST utility is the tool to use.

```
+++TEST                                    <Bad disk is in work drive>
SYSTEM INFO SECTOR INVALID                 <This is to be expected. The
                                            system info sector had not
                                            been created by NEWDISK.>

MAXIMUM TRACK/SECTOR READ: 00/00
ARE THESE ACCEPTABLE? N
HARD DISK OR FLOPPY DISK (H/F): F
DISKETTE SIZE (5/8): 8                     <Assume 8 inch, single-sided,
                                            single-density>

SINGLE OR DOUBLE SIDED (S/D): S
SINGLE OR DOUBLE DENSITY (S/D): S

ADDRESS: 000A, CRC ERROR
TEST COMPLETED
```

The problem is that we have a bad sector in the directory track. If this sector can be removed, and the system information sector built, then the disk would be usable. It would not be the same as if NEWDISK had initialized it, but it would suffice for a scratch disk. It would not be possible to remove this sector if it were the first directory sector (0005); but since it is not, it can be removed with little effort. The first thing to do is outline the exact procedure to be followed.

1) Remove the bad sector from the chain
2) Break off the directory chain from the free chain. Since NEWDISK did not finish, the entire disk is linked together in the free chain. We have to break off the directory from the rest of the free chain.
3) Insert data into the system information sector.

The EXAMINE utility will be used to perform the above operations.

```
+++EXAMINE                                 <The disk is in the work drive>
SYSTEM INFO SECTOR INVALID                 <This is to be expected. The
                                            system info sector had not
                                            been created by NEWDISK.>

MAXIMUM TRACK/SECTOR READ: 00/00
ARE THESE ACCEPTABLE? N
HARD DISK OR FLOPPY DISK (H/F): F
DISKETTE SIZE (5/8): 8                     <Assume 8 inch, single-sided,
                                            single-density>

SINGLE OR DOUBLE SIDED (S/D): S
```

```
SINGLE OR DOUBLE DENSITY (S/D): S
COMMAND: R 9                          <Read in sector 0009>
COMMAND: M                            <Modify starting at byte 0>
00 00 .                               <This byte is unchanged>
01 0A 0B                              <Sets link from, 000A to 000B>
02 00                                 <Carriage return typed>
COMMAND: W                            <Update the sector>
COMMAND: R F                          <Read sector 000F>
COMMAND: M                            <Modify starting at byte 0>
00 01 00                              <Set link to 0000>
01 01 00
02 00                                 <Carriage return typed>
COMMAND: W                            <Update sector. This breaks
                                       the directory from free chain>
COMMAND: R 3                          <Read system info sector>
COMMAND: M 1                          <Modify starting at byte 1>
01 04 00                              <Set link to 0000>
02 00                                 <Carriage return typed>
COMMAND: M 1D                         <Modify starting at byte $1D>
1D 00 01                              <Set start of free chain
1E 00 01                                to 0101>
1F 00 4C                              <Set end of free chain
20 00 0F                                to 4C0F (values for 8 inch
                                        single-sided, single density
                                        diskette>
21 00 04                              <Set free sector count to 1140
22 00 74                               (0474 hexadecimal)>
23 00 .                               <Ignore next 3 bytes>
24 00 .
25 00 .
26 00 4C                              <Set maximum track>
27 00 0F                              <Set maximum sector>
28 00                                 <Carriage return typed>
COMMAND: W                            <Update system info sector>
COMMAND: S
```

Just to be safe, VALIDATE should be run on the disk. If a mistake has been made, VALIDATE will spot it. If the disk had other bad sectors in addition to the one in the directory, FLAW should be run to remove them from the free chain. Note that this disk does not have a boot program on it, so it cannot be used as a system disk.

CASE VI: A Desperate Measure

As we have seen in some of the previous cases, it is possible for data to be damaged such that recovery is impossible. If the data has not actually been overwritten, but is merely "lost" somewhere on the disk, with nothing pointing to it, it may be possible to recover some of it by a rather arduous process. This technique is very time-consuming and should only be used as a last resort to recover extremely important data.

In essence, the technique involves using the EXAMINE utility to dump every sector on the disk, looking for the data. Once it is found, RECOVER can be used to retrieve it. It is obvious that this could take a long time since there are hundreds, even thousands, of sectors on some disks. There are, however, some tricks which can be used to lessen the work involved to a slight degree. If the directory is still intact, the first step is to run FILETEST, to print out all of the known file chains. If you have a printer, this is easy; if you don't, you will have to copy them all down by hand. The command FILETEST +ADFM will print out the chains.

The next step is to determine which sectors have not been listed by FILETEST. This is a manual operation and simply involves looking at the list of sectors that are in known chains, and writing down those sectors that are not in any of the chains. The data is somewhere among these sectors. By using EXAMINE, look at those sectors that are not in any chain. The "C" command is useful for dumping several sectors at a time since the "lost" sectors still form chains amongst themselves. Keep a careful record of those sectors that you have examined, and whether or not they appear to be part of the data for which you are searching. You must be careful since there may be older versions of the data on the disk. Once you have identified the data, you can use RECOVER to copy it to another disk.

As mentioned earlier, this is an extremely arduous process and should be used only as a last resort. Having to go through this process is a painful lesson that could be avoided by having several backup disks of important information.

COMMAND SUMMARY

MEMORY DIAGNOSTICS

```
CONVERGE <starting address>,<ending address>
DYNAMIC <starting address>,<size in 1024 byte blocks>
QUICK <starting address>,<ending address>
RANDOM <starting address>,<ending address>
WALK0 <starting address>,<ending address>
WALK1 <starting address>,<ending address>
```

DISK UTILITIES

```
COPYR <file specification>,<file specification>
EXAMINE <drive number>
     R,<sector address>                 (Read a sector)
     W,<sector address>                 (Write a sector)
     D,<sector address>                 (Dump a sector)
     C,<sector address>                 (Dump sector chain)
     M,<byte number>                    (Modify buffer contents)
     F,<file specification>             (Read first sector of file)
     B,<file specification>             (Build link table for file)
     T,<address>,<address>,<count>      (Move data in memory)
     S                                  (Return to FLEX)

FILETEST <drive number and options>,<file name list>
FLAW <drive number>,<list of sectors>
RAWCOPY <file specification>,<file specification>
REBUILD <drive number>,<drive number>
RECOVER <drive number>,<drive number>
TEST <drive number>
UNDELETE <drive number>
     D  (Dump current file)
     N  (Proceed to Next Older File)
     P  (Go Back to Previous File)
     R  (Recover file)
     S  (Return to FLEX)

VALIDATE <drive number>
```