ORIGINAL ARCHIVER OPERATING INSTRUCTIONS

These instructions describe the operational procedures for the original ARCHIVER only. Installation instructions have been deleted.

2.1.2 Normal Backup Procedure

1. Follow the boot procedure given above (with a copy of the ARCHIVER/EDITOR only).
2. When the ARCHIVER page is displayed (screen changes to a brownish-yellow) then press C (for Copy).
3. The ARCHIVER will respond by asking you to insert source diskette. Now insert the program you wish to backup and then press the START button.
4. After a short time, you will be requested to insert destination diskette. At this time, you should insert the diskette you wish to put the copy on. When you have done this, press the START button.

NOTE

The destination diskette does not have to be previously formatted. The ARCHIVER/EDITOR program formats each track as it is written if the F+ parameter is selected.

5. If the ARCHIVER asks you to insert the source diskette again and repeat steps 3 and 4.
6. Depending on the length of the program, from 1 to 3 passes may be required on a 48K computer. The larger the computer memory is, the fewer the number of passes required. The ARCHIVER will indicate on the screen when the copy is done.
7. We suggest that you put the original diskette away in a safe place and use the backup copy from this time on.

If you got a Read Format Error, most likely you did not follow steps 3 and 4 of the boot procedure carefully. Otherwise the command option parameters may require some changes to enable you to custom modify the diskette copying technique (refer to sections 4.1 and 4.3).

2.2 SOME CONVENTIONS USED

1. All numbers used in the ARCHIVER/EDITOR program are in Hexadecimal (HEX) which is a base 16 numbering system. If you do not understand hexadecimal numbering, then refer to the table in Appendix A. In this manual all HEX numbers are preceded by a $ symbol.

2. In the ARCHIVER/EDITOR program pressing the ESC key will bring you back to the command mode of the program you are currently in. The only exception is during actual disk I/O, (R/W) in which case holding down the OPTION button will stop the disk I/O at the end of the track read/write operation which then allows you to abort the operation by pressing the ESC key or to press START to continue the I/O operation..
3. Whenever disk I/O needs to be performed or continued you must press the START button to proceed.
4. At anytime during the use of the EDITOR program (except during disk I/O) a CRTL-P will create printout of what is currently on the screen on your printer.
5. The CTRL and SHIFT keys need never be used except for printing as described in 4. (However you may press CTRL or SHIFT if you like, but these key functions are disregarded and unnecessary.)
6. Whenever any writing is to be performed the border color will change to red. Whenever any reading is to be performed the border color will change to white.

CHAPTER 3 SCREEN CONVENTIONS

This chapter deals with the various command lines and prompts used by the ARCHIVER/EDITOR program. You should read the following chapters to become aware of all the many capabilities provided by this program.

3.1 ON THE SURFACE

Figure 3-1 shows the screen for the ARCHIVER. However, the EDITOR, the FORMATTER, MAPPER, and the DISASSEMBLER screens all have similar Option, Status, and Command lines. The Option and the Status lines provide 16 unique parameters for disk sector/track format changing. The following paragraphs explain how to use each parameter.

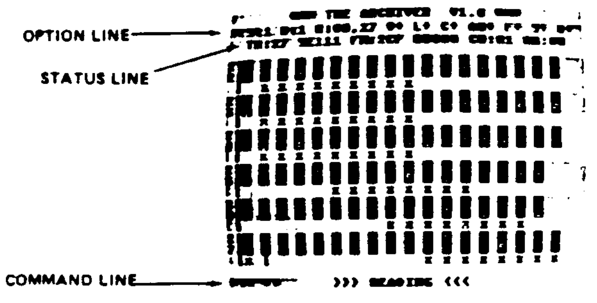


Figure 3-1. Screen Program Lines

3.2 THE OPTION LINE

The option line contains parameters used by the ARCHIVER (and EDITOR). All of these parameters can be changed at any time when you are in the command mode. To modify these parameters type **P** . You will see a cursor on the option line. To move the cursor right and left press the ← or → key (without pressing the **CTRL** key). Pressing **RETURN** selects that parameter to be changed. After the parameter has been changed the cursor will be on the option line ready to select another parameter to change. Pressing the **ESC** key returns control back to the command level. A description of each parameter follows.

3.2.1 **Source Drive**

S:x This is the drive number from which all reading is done. Pressing a **RETURN** when on this parameter will increment the drive number and wrap around at four (4) to one (1). NOTE: This drive must be opened prior to reading from it, otherwise an error will occur (this drive must also have a CHIP installed).

3.2.2 **Destination Drive**

D:x This is the drive number to which all writing is done. Selecting the drive is accomplished the same way as described in section 3.2.1 above.

3.2.3 Track Range

R:xx,yy This is the range of tracks that will be copied using the ARCHIVER (or tracks read/written/formatted... when using the EDITOR). The **xx** is the start track and the **yy** is the end track. When pressing **RETURN** with the cursor positioned on this parameter a prompt will appear on the command line requesting a new range of tracks. There are three allowable syntaxes:

RETURN	:	same as typing 00,27 (tracks 00 to 27 HEX).
x,y	:	set start to **x** and end track to **y**.
x	:	set both start and end tracks to **x**.

ESC will exit this option without modifying the range of tracks. **RETURN** enters the range you entered and updates the option line accordingly. If you make an illegal entry a track range error occurs.

3.2.4 **Verify**

V+ This is the write with verify flag. Pressing a **RETURN** simply toggles this parameter:

+	:	Verify on.
-	:	Verify off.

If the verify is on, a verification will be done on the track after it is written. NOTE: Because the verify pass is separate from the write pass, it is faster than the standard DOS write with verify.

3.2.5 **Logic Seeking Read/Write**

L+ This is the read/write logic seeking flag. Hitting a **RETURN** simply toggles this parameter:

+	:	Logic seeking on.
-	:	Logic seeking off.

When reading or writing multiple sectors with the same number (i.e. two sector $09) you must be able to read or write the correct sector, therefore, there are logic seeking read/write commands in the CHIP that automatically synchronize to the format on the track and read/write the correct sector. Since synchronizing to a track takes a little more than one revolution, these commands are slower than the standard read/write commands. The only time you would want to change this to a - is when the format cannot be synchronized (see section 8.1). If the logic seeking is off, it is suggested that you turn compaction off (refer to section 3.2.6). Note: The ARCHIVER/EDITOR programs only use the logic seeking commands (if enabled) when a non-unique numbered sector is to be read or written.

3.2.6 Compaction

C+ This is the compaction flag. Simply pressing **RETURN** toggles this parameter:

+	:	Compaction on.
-	:	Compaction off.

If you have compaction on when using the ARCHIVER, the sector will neither be read nor written by the ARCHIVER/EDITOR if it is filled by a single value (i.e. $00 etc.). If you are in the L- mode you should have compaction off. Sectors filled with the values $01-$08 will not be compacted as these are format control bytes. These "fill" bytes are placed in the sector automatically when the track is formatted.

The C +/- parameter has the same function in the EDITOR as it does in the ARCHIVER, however, in the EDITOR the results are more readily apparent. Compaction only works on sectors which are not bad and that have single byte filling the entire sector. Also, sectors filled with the values of $01-$08 will not be compacted. If the sector was compacted, the EDITOR will NOT display the data in the sector. The EDITOR will only display sectors it actually read. The CHIP actually reads the data and reports back to the EDITOR that the sector is to be compacted, thus saving time on reading a diskette.

3.2.7 Format Read Type

A6+ This is the type of track reading that the ARCHIVER/EDITOR program will use to determine the format on the tracks. Either 4 or 6 bytes of information about the sector can be selected (A4 or A6). The + or - is the toggle to turn on (+) or off (-) the format verification logic. Normally the **A6+** will be desired. To change this parameter, simply press **RETURN** with the control cursor positioned on the **A6** . The meaning of each of the codes is as follows:

- **6** : Six bytes are returned to the ARCHIVER/EDITOR for each sector, thus the ARCHIVER/EDITOR will be able to rotate the sequence so that the end-of-track gaps will be identical (**A6+** only). This is mainly cosmetic, but does have significance on fast formats.
  Because 6 bytes are returned, a maximum of 21 sectors per track can be fetched. If there are more than 21 sectors, then a **4** mode should be used.
- **4** : Four bytes are returned to the ARCHIVER/EDITOR for each sector, thus some information about each sector is missing. This is intended for 22 to 24 sector formats.
- **+** : The track is cycled through twice comparing the first sector sequence to what the CHIP finds the second time. This is an internal function of the CHIP.
- **-** : This mode is slightly faster than the + mode, however, no verify is done on reading the format. This is generally used for speed and also if the track is badly garbled. (Unformatted tracks can return strange sector headers on some diskettes.)

For more information on the difference on the 6/4 byte read distinction, see section 5.15.

3.2.8 Format Flag

F+ This is the format before write flag. Normally you will want a **F+** mode. Simply pressing **RETURN** will toggle this flag when the cursor is positioned on the **F+** .

- **+** : Format track before doing the write pass.
- **-** : Do not format. This option is selected if you already have an identical format on the track or if you are simply trying to put sectors on the destination track. If there are multiple sectors with the same number and the track formats are not identical the logic seeking read/write commands will not work correctly. Also, the verify may not work correctly if it tries to verify the wrong sector. This flag also allows you to convert slow formats by first formatting the destination track with a fast format and then write out the sectors that were read from a slow formatted diskette.

3.2.9 Screen Code Conversion

S+ This is used in the EDITOR only. It refers to the conversion of characters displayed on the normal EDITOR page to the right of the sector display. A **RETURN** toggles this parameter.

- **+** : Convert data to ATASCII characters.
- **-** : No conversion. Display data as Atari screen codes.

3.2.10 Bad Sector (CRC)

B+ This flag refers to the method of writing CRC bad sectors. Pressing RETURN toggles flag on (+) or off (-). This flag should always be set to + when in the ARCHIVER.

- **+** : Write a full bad CRC sector.
- **-** : Only write a partial sector (CRC bad). The number of bytes written depends on the last byte of the sector data. That byte refers to the number of bytes that will be written. This allows for the capability of increasing the number of sectors on a track to above 20 (i.e. two half sectors take about the same amount of room as a full sector).

3.3 THE STATUS LINE

The status line is the third line on the screen. It will display the current track, sector, composite sector number, the amount of free buffer memory, current copy number and the number of copies to make (in the ARCHIVER or the sector data address in the EDITOR). The only directly adjustable parameters are the CO:xx which refers to the number of copies to make and the LOC:xxxx which is the sector start address. The status line parameters are as follows:

TR:xx This is the current track number the ARCHIVER/EDITOR is processing. (Tracks range from $00-$27.)

SE:xx This is the current sector number the ARCHIVER/EDITOR is processing. (Sectors range from $01-$12, a — means that the sector number is invalid.)

FM:xxx This is the composite sector number used by Atari DOS. These numbers are arrived at by the formula **FM = TR*$12+SE.** Where **TR** is the track number and SE is the sector number. The FM ranges from $001 to $2D0. A — indicates that the sector number is invalid.

$xxxx This is the current free memory for storage of the sector data and track information. When data is being read into the buffers, the memory counter will decrement $80 for each sector read and also for each track read. NOTE: If compaction is on, compacted sectors do not take up memory space, however, there is a $80 byte overhead to store sector layouts and various other information for each track. On a 48K machine this field will read $9900 (about 38K).

NU:xx	This is the number of the copy being made. A $00 indicates it is on a read pass. A $01 to $FF is the number of the current copy being written.
CO:xx	This is the number of copies to be made per each read pass. This is defaulted to one ($01) whenever the ARCHIVER program mode is entered. This value can range from $01 to $FF.
LOC:xxxx	This parameter is used with the EDITOR and is the address location in which all disassembly or displays of sector data will start. This is for purely comestic reasons and does not affect the data (refer to section 5.12).

3.4 THE COMMAND LINE

The command line is at the bottom of the display. This line will contain all necessary screen prompts, input commands and error messages. When using one key command entries no **RETURN** is necessary to enable that command. Simply press the desired key for the desired command input. However, on numeric input pressing **RETURN** is necessary to enter the numeric information.

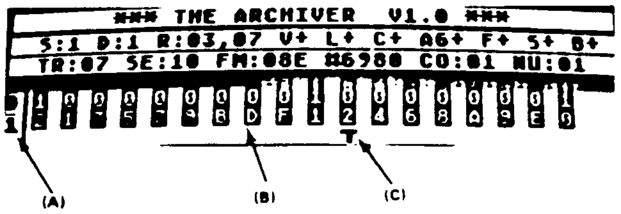
Also, pressing the space bar will erase an error message or copy done/aborted message immediately. Otherwise the message will disappear after approximately 4 seconds.

3.5 OPENING/CLOSING THE CHIP

Normally the CHIP will already be open if the Disk Drive is booted correctly (refer to section 2.1). However, there may be some cases in which you will need to open a drive. (Opening a second drive for example or if the drive was not booted correctly.) To open the CHIP, type an **O** when in the command mode (in either the ARCHIVER or EDITOR). You will be prompted to enter the open code and drive number. Enter your code, and the drive number (optional—the default is one). If you enter a wrong code or just press **RETURN**, the CHIP will close. Pressing **ESC** aborts this option.

3.6 SECTOR DISPLAY FORMAT

The ARCHIVER/EDITOR's sector layout displayed on the screen is somewhat unique. Field (a) (shown in figure 3.2) is the track number (HEX) from where the sector sequence came. The numbers in field (b) represent the actual sector numbers on the track and are in the sequence as found on that track. The numbers are read vertically (i.e. figure 3.2 shows track = $01, sectors = $12, $01,...). Generally there will be $12 sectors (18 decimal) on a track. However, this can vary from one software protection scheme to another. Field (c) represents the status of the sector. If there is a symbol under the sector number, the sector is considered 'bad' and will return a bad sector status if read (a protection technique). Refer to table 5.1 in section 5.10 and to section 6.11 for each symbol's meaning. The sector numbers can be in any order and need not be unique. Two (or more) reads of the same sector number need not return the same data.



```
*** THE ARCHIVER  V1.0 ***
S:1 D:1 R:03,07 V+ L+ C+ A6+ F+ S+ B+
TR:07 SE:10 FM:08E #6980 CO:01 NU:01
```

Figure 3.2 Track/Sector Display Format

For detailed information on the source of these sector numbers, refer to the section on the track layout (section 6.3) and the following paragraph.

These track and sector numbers are not used internally by the Atari computer. Instead, the operating system refers to each sector as a number from $001-$2D0 (1-720 decimal). The computer's disk operating system (or DOS) will access the Disk Drive using this composite sector number. Then, within the Disk Drive, the composite sector number is broken down into a track and sector number using the relationship:

composite = (track) * ($12) + (sector)

Thus, the first sector in figure 3.2 ($12) would be called $24 (36 in decimal) within the computer. Notice in the figure that there are two sectors with the number $09. If the Atari computer were to read sector $2B (composite remember), it would get one of the two possible sectors. This is called a 'double sector'.

CHAPTER 4
THE ARCHIVER

The ARCHIVER is an automatic copier designed to copy your protected (or unprotected) software for backup purposes. The ARCHIVER is easy to use and will backup virtually all protected software.

4.1 AN OVERVIEW

In general, diskettes can be copied by simply typing a C . For some special disk formats it may be desirable to change several of the ARCHIVER operating parameters. The ARCHIVER will allow the making of multiple copies per each read pass. On a 48K system a disk will take up to 3 passes to copy. However, most diskettes can be copied in one or two passes depending on the amount of data on the diskette.

As a safety feature the ARCHIVER/EDITOR requires that you press the **START** button before any disk reading or writing will take place. If you wish to abort the reading or writing during disk I/O press the **OPTION** button and hold it down until the track is completely read or written. To continue press the **START** button and to exit the operation press the **ESC** key. The **ESC** key will always return control to the previous command mode while disk I/O is non-active.

4.2 NUMBER OF COPIES

This command will allow you to select the number of copies that will be made on each read pass. To enter the number of copies you wish to make, type an **N** . You will be prompted to enter the number of copies to make. Type the number (in HEX) followed by a **RETURN** . The number selected will be reflected after the **CO** . When making copies on a single drive, screen prompts will signal when to insert the source diskette and when to insert the destination diskette. On a two drive system (both with a CHIP), the first copy will be made automatically and subsequent copies will be prompted. The number after the **NU** indicates which copy is currently being processed. A $00 means you are on the read pass.

4.3 AUTOMATIC COPY

The command to start making copies is initiated by pressing the **C** key. When activated, screen prompts will be displayed for inserting the source (original) and destination (backup copy) diskettes throughout the process. Remember to press **START** to acknowledge to the prompt that you are ready. The copy command **C** makes the number of copies specified by the **CO:xx** field and does its functions according to the parameters on the option line (if applicable). The memory buffer containing the previously read data will be cleared prior to each read pass.

If you have problems copying, check the following:

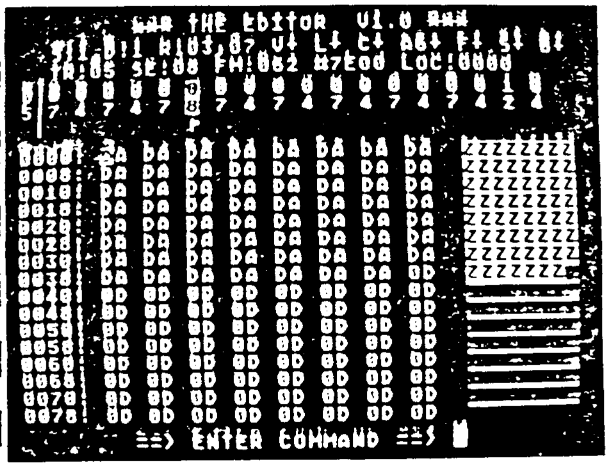
1. Change to a different destination diskette.
2. A6+ to A6-.
3. A6, L, and C to -.
4. If the diskette has 20 or more sectors on a track, then read each sector/track using the Editor and write it onto the destination diskette. Refer to sections 6.11 and 8.2.
5. Be sure you have a data separator board and that the disk drive is running at the right speed.

4.4 ENTER EDITOR

To enter the EDITOR type **E** . All data currently in the memory buffers will transfer.

CHAPTER 5
THE EDITOR

The EDITOR will allow you to actually edit the sector data and do many manipulations with it. Custom formatting can also be done, thus enabling you to make protection schemes or modify protection schemes as desired. Because formats can now have over 19 sectors, the EDITOR is necessary in order to duplicate these sophisticated formats. (Formats greater than 19 sectors have never been used to protect diskettes designed for use on the Atari computers before the introduction of the CHIP.)



```
        *** THE EDITOR  V1.0 ***
  [illegible] [illegible] R:03,07 V↓ L↓ C↓ AB↓ F↓ S↓ B↓
   TR:05  SE:08  FM:062 #7E0D  LOC:0000
 0|0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
 5|7 4 7 4 7 8 7 4 7 4 7 4 7 4 7 4 2 4
0000: DA DA DA DA DA DA DA DA   ZZZZZZZZ
0008: DA DA DA DA DA DA DA DA   ZZZZZZZZ
0010: DA DA DA DA DA DA DA DA   ZZZZZZZZ
0018: DA DA DA DA DA DA DA DA   ZZZZZZZZ
0020: DA DA DA DA DA DA DA DA   ZZZZZZZZ
0028: DA DA DA DA DA DA DA DA   ZZZZZZZZ
0030: DA DA DA DA DA DA DA DA   ZZZZZZZZ
0038: DA DA DA DA DA DA DA 0D   ZZZZZZZ
0040: 0D 0D 0D 0D 0D 0D 0D 0D
0048: 0D 0D 0D 0D 0D 0D 0D 0D
0050: 0D 0D 0D 0D 0D 0D 0D 0D
0058: 0D 0D 0D 0D 0D 0D 0D 0D
0060: 0D 0D 0D 0D 0D 0D 0D 0D
0068: 0D 0D 0D 0D 0D 0D 0D 0D
0070: 0D 0D 0D 0D 0D 0D 0D 0D
0078: 0D 0D 0D 0D 0D 0D 0D 0D
        ==> ENTER COMMAND <==
```

Figure 5.1. EDITOR Screen

5.1 AN OVERVIEW

The EDITOR is designed to be easy to use yet it doesn't lack in sophistication. One key commands allow you to browse through the many parts of the EDITOR. Unlike the ARCHIVER, only one track's sector list will be displayed at a time. The EDITOR allows you to move between sectors by simply pressing the left and right arrow keys (← and →). You will notice the dual purpose of the track format lines as both a sector selection aid and as a sector layout display. This will be discussed in more detail later. The normal EDITOR display will be of the actual sector data of the sector that the cursor is on (on the sector layout lines). If there is no track in memory, the sector layout lines will be blank.

The main sector data display will contain data only if there is at least one track in the memory buffer and the sector that the cursor is on contains data.

5.2 READING TRACKS

To read in a range of tracks first be sure that the **R:xx,yy** parameter is correct, then type an R followed by pressing the **START** button to start the read process. As a safety feature, if a track is currently in memory that was specified in a read operation, the reading of that particular track will not occur. That track will be skipped and the read process will continue with the next track.

5.3 WRITING TRACKS

To write a range of tracks first set the track range (as in the read). Press **W** along with **START** to initiate the writing process. Only the tracks and sectors actually in memory within the range selected will be written. If formatting is to occur before the write, the fill bytes will be written during the format on compacted sectors. If a sector was deleted that sector will not be written. If formatting is on then zeros will fill that sector.

5.4 ENTER EDIT MODE

Prior to entering the Edit Mode, the sector data must first be displayed. If so, press **E** to enter the Edit Mode. Otherwise, read in the track you want to edit, then press E . The cursor appears within the sector data and you may start editing the code. The commands available for use while in the Edit Mode are as follows:

←	:	Move cursor one byte toward the beginning of the buffer (left).
→	:	Move cursor one byte toward the end of the buffer (right).
↑	:	Move cursor eight bytes toward the beginning of the buffer (one line up).
↓	:	Move cursor eight bytes toward the end of the buffer (down one line).

RETURN :	Move the cursor to the beginning of the next data line.
DELETE :	Delete the byte the cursor is on. All data beyond the cursor moves up one byte and a zero is placed in the last byte of the sector.
INSERT :	Insert a byte at the cursor position. All data moves down one byte from the data that the cursor was on. The last byte of the buffer is lost.
CLEAR :	Fill the entire buffer with the character currently under the cursor.
H :	Move the cursor to the first byte in the buffer.
xx :	Typing HEX numbers changes the data to exactly what you see. The cursor will automatically move to the next byte when a byte has been entered. All spaces are automatically skipped between each byte.
ESC :	Exit the edit mode. All changes will be saved to a memory buffer (not the disk) and are permanent unless changed later. This will also update the characters on the right to their new value. (This is not done automatically during the Edit Mode.)

The address at the left is arbitrary and is used strictly for reference. The address can be changed by the L command (see section 5.12).

5.5 DISASSEMBLY

The EDITOR has a built in disassembler. First enter the Edit mode and then move the edit cursor to the byte at which you wish to begin the disassembly. Exit the Edit mode (press ESC) and then press D to begin the disassembly. The disassembled listing will instantly be displayed on the screen. To scroll up or down the listing press the up (↑) or down (↓) arrows. The disassembly will not scroll above the byte that the edit cursor was on and the disassembly will not proceed beyond the end of the sector. Scrolling will occur in increments of eight lines. To exit the disassembler, press the ESC key. Pressing CTRL-P will dump the screen to a printer if desired.

5.6 MOVEMENT BETWEEN SECTORS

When in the command mode the cursor movement keys allow you to move from one sector to the next. The right (→) and left (←) arrow keys will move the sector cursor right and left. This allows you to display any sector in that track. The up (↑) and down (↓) keys moves the Edit display screen between tracks. If the track is in memory that track will be displayed, otherwise, that track will be skipped and the next track present will be displayed. If the cursor happens to rest upon a sector which is not in memory the sector data window will be blank. Sectors which have an x under them cannot be viewed. This is because these sectors are inaccessable to a normal 810 Disk Drive. As you move from sector to sector, the track, sector, and composite numbers are automatically updated.

5.7 CLEAR TRACK FROM BUFFER

The CLEAR key will delete an entire track from memory. The next track will then be displayed. The memory indicator will automatically be incremented reflecting the deletion. If you wish to delete all tracks from memory, simply holding down the CLEAR key will do the job. Pressing RESET also clears tracks from memory, but it sets all parameters to their default values.

5.8 CLEAR SECTOR FROM BUFFER

The DELETE key will delete the sector currently displayed. If no sector is being displayed, a beep will sound to indicate that there is nothing to delete. If a write occurs, that sector's data will not be written, however, the sector header will be put on the diskette (if formatting is on). Deleting a sector simply erases the data and does not modify the track layout.

5.9 TRANSFERRING SECTORS

Typing an H will copy the sector being displayed into a hold buffer. Pressing the INSERT key will copy the buffer to the sector the cursor is currently on. If a sector is being displayed the new data will simply replace the old. If the sector was originally empty the new data will simply be inserted. NOTE: All disk I/O uses the same buffer so the data held will be lost.

5.10 CREATING BAD SECTORS

When a sector is being displayed you can cause that sector to be bad by pressing the B . When you do this, only a flag is changed so you must write the entire track in order for the sectors to be written as bad. If there is no data in the sector the sector will not be written. Thus that sector will not be bad on the track. ONLY SECTORS ACTUALLY WRITTEN WILL BE BAD (if they were selected to be bad). There are seven types of bad sectors possible using this method (see table 5.1). There are three flags that can flag a bad sector. Any combination of these three flags can be set by pressing B . The symbol under the sector number will cycle through all combinations of bad sectors plus one of good sector. The reason for having several types of bad sectors is that the three flags mentioned above can each be read and examined on an unmodified 810 Disk Drive.

SYMBOL	BIT 6	BIT 5	BIT 3	
┌	CLR	SET	CLR	BIT 3 : CRC error bit. BIT 5 : Data type flag #1. BIT 6 : Data type flag #2.
┐	SET	CLR	CLR	
┬	SET	SET	CLR	
│	CLR	CLR	SET	
├	CLR	SET	SET	
┤	SET	CLR	SET	
┼	SET	SET	SET	
(blank)	CLR	CLR	CLR	

Table 5.1. Types of Bad Sector Symbols

When you press the **B** key the symbols cycle through in the order as shown above. Only the last entry is a good sector.

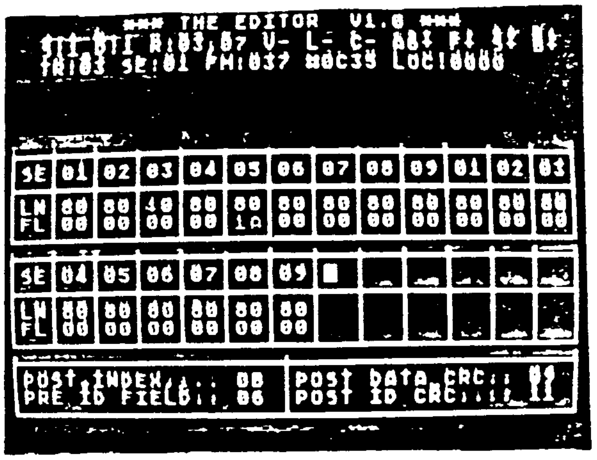
NOTE

These bit numbers refer to the status byte returned when executing a STATUS COMMAND (not the I/O status returned after the read).

5.11 CUSTOM FORMATTER

The Custom Formatter allows you to create your own sector layouts and format a range of tracks using your own layout. You can create any sequence of sector numbers you desire. The only restriction is that only sectors with numbers between 1 and 18 can be read.

To enter the Formatter type **F** . The Formatter has its own screen layout which allows you to set the formatting parameters (except for the range) in which you would like to format. Thus, before entering the formatter, you should select the range of tracks to format from the EDITOR.



```
          *** THE EDITOR  V1.0 ***
  [illegible] R:03,07 V- L- C- [illegible]
  TR:03 SE:01 FM:037 #0C39 LOC:0000

SE 01 02 03 04 05 06 07 08 09 01 02 03
LN 80 80 40 80 80 80 80 80 80 80 80 80
FL 00 00 00 00 1A 00 00 00 00 00 00 00

SE 04 05 06 07 08 09
LN 80 80 80 80 80 80
FL 00 00 00 00 00 00

POST INDEX..: 0B    POST DATA CRC:: [illegible]
PRE ID FIELD:: 06   POST ID CRC::: [illegible]
```

Figure 5-2. Formatter Track Layout

The **SE** row contains the sector numbers which will be placed in the headers of the track (refer to figure 5-2). The **LN** row contains the number of bytes that will be in the sector data and the **FL** row contains the data fill byte that will go into that particular sector. NOTE: Fill bytes of 1 to 8 must not be used as these bytes have special signficance to the disk drive FDC circuit during formatting. Sector $03, for example, will only contain $40 bytes (64 decimal) and if read will return a bad status. Sector $05 will contain the normal number of bytes, $80 (128 decimal) but will be filled with all $1A. There are two tables of twelve sectors each in the formatter screen layout page. They should be considered sequential (there wasn't enough room to fit 24 sectors on one row!) The table below the sector tables contains the gap length bytes.

Because a track is only so long only a limited number of bytes can be placed on a track. After the **#** is the current number of bytes the formatter has calculated your format will use on the track. This number must remain between $BCO and $CBO for your format to be reliable.

All editing changes in the formatter will remain intact until you reboot the ARCHIVER/EDITOR diskette. No defaults are stored back in this table. Therefore, you can go back and forth between the edit page and the format page without loss of the new format.

The commands used in the Formatter are:

←	:	Move cursor left one sector (or gap size value).
→	:	Move cursor right one sector (or gap size value).
↑	:	Move cursor up one parameter field (i.e. FL - LN - SE - gap values - FL . . .).
↓	:	Move cursor down one parameter field.
DELETE	:	Delete sector cursor is on or if the cursor is past the last sector, delete the last sector.
INSERT	:	Insert a sector before the sector that the cursor is on.
CLEAR	:	Clear entire format (start from scratch).
xy	:	Hex entry overwrites what is currently dislayed.
ESC	:	Exit; go back to the Edit screen.
W	:	Format the range of tracks (**R:x,y**) using the format created.

5.12 Address Changing

The address at which the sector begins may be changed by pressing the L key. Answer the prompt by entering the new address in hexadecimal. This address is used only as a reference and does not physically relocate the buffer contents.

5.13 INSERTING CUSTOM FORMAT

Pressing the I key allows the insertion of custom formats from the Formatter page into a range of tracks (Rxx,yy). The old tracks (if any) will be replaced. No sector data will transfer. To insert data in the new sectors, you must use the H and INSERT keys.

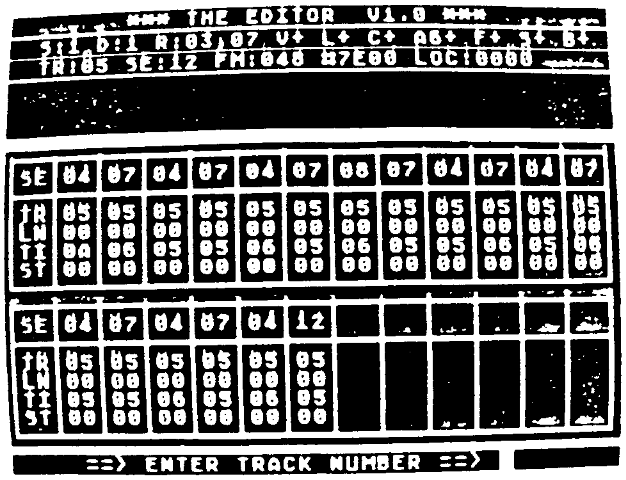
5.14 MOVING TRACKS

Tracks can be moved (but not duplicated) by pressing the N key. The track currently displayed will be renumbered to a new track number that you enter. The track currently at the destination spot will be deleted and the track you are on will be deleted from its current place and be moved to the new location.

5.15 TRACK MAPPER

Pressing an M is used for entering the Mapper page. This function will allow you to examine the format of individual tracks. The most significant function of this command is to allow you to determine the gap size between successive sectors.

The SE is the sector number that originates from the sector header. (Refer to figure 5-3.) The TR is the track number as found in the sector header, and the LN is the sector length byte. For more information on these values, refer to section 6.3. The TI is the amount of time between that sector and the succeeding sector in units of 2048 (decimal) microseconds. There are about 100 (decimal) units of time on a track, so the sum of these numbers should be about 100.



```
            *** THE EDITOR  V1.0 ***
 S:1 D:1 R:03,07 V+ L+ C+ A6+ F+ S+ B+
 TR:05 SE:12 FM:048 $7E00 LOC:0000

SE 04 07 04 07 04 07 08 07 04 07 04 07
TR 05 05 05 05 05 05 05 05 05 05 05 05
LN 00 00 00 00 00 00 00 00 00 00 00 00
TI 0A 06 05 05 06 05 06 05 05 06 05 06
ST 00 00 00 00 00 00 00 00 00 00 00 00

SE 04 07 04 07 04 12
TR 05 05 05 05 05 05
LN 00 00 00 00 00 00
TI 05 05 06 05 06 05
ST 00 00 00 00 00 00

   ==> ENTER TRACK NUMBER ==>
```

Figure 5-3. Track Map Layout

The ST is the status of the sector header read. Anything other than a zero means that the sector can not ever be accessed. Also, any A4 read format mode will not return the TI and ST values. This is because the A4 mode goes for quantity as far as sectors go, while the A6 modes for quality of information per sector.

NOTE

The last sector's TI (time) value will only be correct on an A6+ read format mode.

5.16 ENTER THE ARCHIVER

To enter the ARCHIVER from the EDITOR you must type an A . CAUTION: all data currently in the data buffers will be lost as soon as the ARCHIVER command C is used. However, the data will not be lost if you immediately return to the EDITOR.

CHAPTER 6
DISK FORMATTING THEORY

By far the most powerful feature of the CHIP over the Atari C ROM is its ability to create custom formats and successfully write (and read) sectors of these formats. By no means do we expect you to fully understand the peculiarities of disk formatting and general I/O with one reading. Remember, it took a couple of years for software houses to devise even the simplist of protection schemes, so don't expect to learn it all in an hour. However, we feel that to use the ARCHIVER/EDITOR to its fullest, at least some basics should be understood. In this chapter, the very basics will be presented, and gradually the specifics of the track layout and protection schemes will be dealt with.

6.1 AN OVERVIEW

The Atari 810 Disk Drive is an intelligent drive which means it is just another computer, capable of reading and writing diskettes and relaying the information to and from the main computer. The CHIP is just a program much like the Atari OS that adds a wide variety of functions to the 810 Disk Drive. A description of the commands understood by the old ROM C and the operation of the SIO is given in the Atari OS manual so it will not be repeated here. For the remainder of this chapter, only the workings of the disk drive and the CHIP will be considered, so it is assumed that you know the theory of communication between the computer and the disk drive.

6.2 DISKETTE STRUCTURE

A diskette is composed of a thin magnetic disk covered by an outer rigid black cover. The outer cover (or jacket) has an oval open area on both sides exposing the disk surface to the drive read/write head. As the diskette spins about its central hub while inside the drive, the read/write head hovers over the jacket oval opening and reads the disk surface much like a cassette recorder would.

The diskette is electromagnetically divided into 40 tracks. A track is a ring about the center of the diskette. The disk drive's head can be positioned precisely over any one of the 40 tracks, thus data can be sequentially read in as the disk surface spins underneath the head as in a cassette recorder.

The track data magnetic fields are converted into electric pulses which are fed to the FDC (floppy disk controller). The FDC is the interface between the read/write head and the drive's microprocessor. The FDC is responsible for interpretting and processing commands from the microprocessor. The FDC performs all sector searches and is an intermediary on all sector data transfers between the microprocessor and the physical disk surface.

Because each track contains too much data that must be handled for each revolution of the diskette a subdivision of the track is necessary. Thus, the track is normally divided into 18 sequential sectors of $80 (128) bytes of data each. Besides being easier to deal with, error checking and reliability are not much of a problem. As you may be aware, all the protection schemes deal with the sector in one form or another, so the rest of this chapter will deal explicitly with the sector.

6.3 THE BASICS OF A SECTOR

A sector has two parts to it; the header and the data. Because the track is circular, there is no way to distinguish the beginning of a track from the middle, thus, a sector needs to be able to identify itself to the controller. This is the purpose of the sector header. These sector headers are written during formatting, so the sector can be identified upon subsequent reading and writing to and from the sector.

Figure 6-1 shows the typical 810 sector/track layout format and the following paragraphs describe the various contents that make up the sectors.

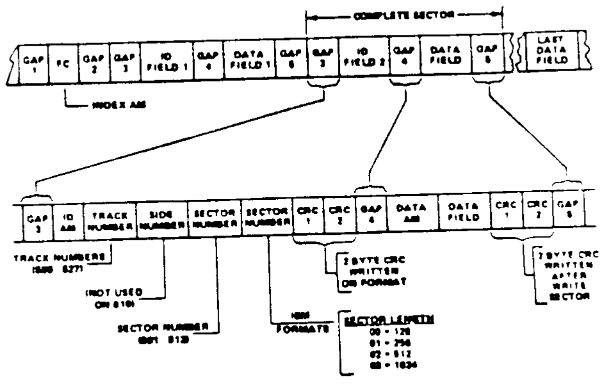


Figure 6-1. Sector/Track Format

6.4 TRACK LAYOUT/FORMAT

Disk formatting is accomplished by the write track command. Each byte for the entire track must be provided for proper formatting including the gaps as well.

The FDC requests each byte in turn and places it directly onto the surface of the diskette. However, there are exceptions to the rule. If data bytes $F5 through $FE are fed to the FDC, it recognizes these as special control bytes and take appropriate action. The byte sequence is illustrated in figure 6-1.

Gap size restrictions:

GAP 1 :	This is always 255 ($FF) bytes and may be over-written by the last sector on the track. This is to ensure that no garbage remains between the last sector and the first.
GAP 2 :	(Post Index AM gap) This gap should be at least one (1) byte.
GAP 3 :	(Pre ID AM gap) This gap should be at least one byte.
GAP 4 :	(Post ID CRC gap) This gap must be $11 (17) bytes in length. (See Read/Write sections.)
GAP 5 :	(Post DATA CRC gap) This gap should be at least one, however, in practice, it should be over 9 bytes long. This is to protect the next sector header from being overwritten.

6.5 THE READ COMMAND

When the processor issues the read command to the FDC, a search for the sector header begins. The FDC reads the headers of the sectors it finds and compares the sector number and the track number to those given by the processor. If the test fails, the search continues. Next, the CRC is checked for validity; if not correct, the search continues. If all is correct, the FDC begins searching for the data AM. If found within 28 bytes, the sector is read byte by byte and is transferred to the processor. Finally, the CRC is checked for validity at the end. The CRC status error bit is set accordingly. Also, the type of data AM byte will determine the status' of bits 5 and 6 of the status register. If the sector is never found ie. ID fields don't match, bit 4 of the status is set, and the processor (CHIP) will reposition the head in hope that somehow the head had gotten over the wrong track (grind!!), and try again.

6.6 THE WRITE COMMAND

This works identically to the read command except that once the sector has been located, a write occurs. NOTE: The write requires that $11 (17) gap bytes be between the sector header and the data. Also, the data AM byte's value depends upon the last two bits of the write command byte. On three of the four possibilities, the processor will interpret the sector as 'bad' (see section 6.11).

6.7 THE CHIP'S LOGIC SEEKING READ/WRITE COMMANDS

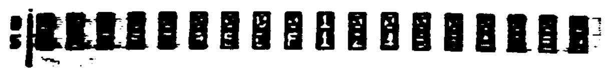
These are the read and write commands that are used for double sectors. The CHIP will first compare the sector sequence it contains to what it finds on the diskette. When it syncronizes itself to the sequence, the write or read function described in section 6.5 and 6.6 will take place. The CHIP is able to get the sector headers through a read address command (of the FDC) which returns the six bytes contained in the sector header (track,...,CRC bytes).

6.8 READ FORMAT COMMANDS

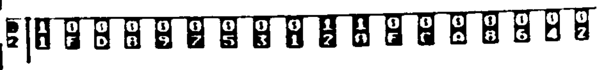
Using the method described above, the sector sequence can be fetched. On the A+ modes, the headers are continuously read for slightly more than one revolution. After this, the sector numbers are compared on the next revolution and the first sequence is cropped to agree with what it finds the second time through. The A- modes read for about one revolution but no double check is made.

6.9 SIO SPEED RESTRICTIONS

The disk drive's processor (and therefore the FDC) receives a full sector of data every 1/18 of a disk revolution. This is about .0115 second, however, the serial transfer between the computer and the disk drive is considerably slower, (about .09 second). Now, since the diskette is turning at 288 RPM (or 4.8 rpms), if you do a little math, you will find that only two sectors can be read in one disk revolution. This is the concept behind fast formats.



Above is the standard format used in the CHIP as well as the Atari ROM C. Notice that consequetive numbered sectors are nine apart within the sequence and ten apart when crossing the end of track gap (which is about half a sector in length). If you are thinking ahead you may realize that even this format can be improved upon.



In the above format, the sequential sectors are nine apart except for the end of track gap, in which case they are eight apart. Here, that gap is large enough such that the eighth can just be read before the head passes it by (or rather it passes the head by). This format is the fastest format possible on the 810 disk.

6.10 DOUBLE SECTORS

Now suppose that two sectors had the same number. If you just randomly went and read that numbered sector, you could get two different sets of data. This process can be precisely controlled by first reading the sector nine (9) places before the one you really wish to read, and then read the one you want.

```
0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
5 4 6 5 7 6 8 7 9 8 2 9 2 1 3 2 4 3

a b c d e f g h i j k l m n o p q r
```

The above sector sequence contains 18 unique sectors but 8 numbers are duplicated. (This is actually a format used in the protection of one software house.) Now suppose you read sectors in the following order:

12, 4, 9, 5, 3, 9.

The actual physical sectors would be as follows:

k, b, l, d, o, i.

You will notice that the two reads of sector 9 did not yield the same result, thus this becomes a valid protection scheme. This is a rather new protection method (mid 1982), yet it is simple to understand and to duplicate (with the CHIP). This type of protection can ONLY be created with a drive modification (which is exactly how they are created in the first place).

This idea can easily be expanded upon to include triple or quadrouple sectors. HOWEVER, the ability to consistantly and reliably get the same results gets harder with the more duplicate numbered sectors you have. Another application is to create more than 18 sectors and number two with the same number. Previously, this was difficult to grasp and realize the feasibility of such a scheme, however, now with the EDITOR, you may create as many as 24 sectors on a track, but because there is only so much room, many sectors must be cut short (and thus be bad sectors). A word of warning: the data in short sectors is not always reliable and timing between sectors is not the same. Timing becomes critical in importance and slight variations in speed may have adverse effects on protections.

6.11 BAD SECTORS

The ability to write bad sectors has been around for quite a while now. It was the first type of true protection, but is now becoming not so important. It is possible to create two types of bad sectors with a standard 810 Disk Drive. The first is a CRC error and the second is a missing sector. The CRC error bad sectors were created by one of two methods; the first being slowing down the drive, and the second being the tape method. The missing sector was created by writing to the preceeding sector at a high RPM, thus causing the end of the first sector to overwrite the header of the next.

Now, creating bad sectors is an easy and valuable function of the CHIP. To create a missing sector, simply format the track without that sector number. To create CRC bad sectors, special operations must be performed by the CHIP while writing the sector. These functions are all automatic and easy using the ARCHIVER/EDITOR, however, a brief description of each type will be given below.

6.11.1 CRC Error Sectors

The CRC bytes are a sophisticated checksum of the preceeding data in a sector. If these bytes do not agree with the data read from the sector, a CRC error will occur. This type of bad sector is simply created by stopping the write process in midstream, thereby keeping the old CRC yet allowing new data. The status CRC error bit (bit 3 of the status) will reflect the error after a read. The CHIP also carries this process a step further. You can specify the number of bytes actually written when creating a bad sector by putting the number of bytes to be written in the last byte of the sector data. After the last byte is written, the process stops, and on subsequent reads of that sector, the status will reflect a CRC error (on the B- mode only).

6.11.2 Data Type Flags

Another way to create perfectly good sectors with a bad status is by setting data type flags in the write (FDC write) command. When this is done, the data AM mark bits 0 and 1 are changed to reflect the type of data. Although these sectors are perfectly good, the CHIP (and the ROM C) will take these sectors as being bad and return an error. Bits 5 and 6 of the status will reflect the results of the read of these types of sectors. With two bits, four combinations can be made; only one of which is a perfectly good sector.

In all there are nine types of sectors: Only one of which is good. The missing sector is another type and the remaining seven are created by combinations of the data type flags and the CRC error bit.

6.12 STATUS

The bits referred to as being status bits 3-6 are not automatically had after reading a sector. The meaning of the SIO status is as follows:

$90 :	A bad sector of ANY type was encountered upon the read.
$8A :	Timeout. The sector was missing and the drive did not respond in time.
$8B :	Device NAK. related to above. If the drive doesn't respond in time, the SIO tries again, however.
$8C :	Serial bus. Related to above.
$01 :	A good read/write

The $90 should usually be returned on bad sectors, however, the timeout value of the disk interface routine is borderline thus causing the errors $8A-$8C. A $90 can be insured by setting the timeout value higher and using the SIO instead.

The status bits of the FDC are received by executing an S (status) command after reading the sector in question. The S command will return 4 bytes of which only two are really meaningful and only the second is described here. For reference to the others, see chapter 5 (Diskette Handler Commands) of the Atari OS manual. After a read, the hardware status bits are reflected as in figure 6-2.

BIT	READ	WRITE	NOTES
7	Not ready	Not ready	always CLR
6	Data type	Write protect	
5	Data type (a)	Write fault	
4	Record not found	Record not found	(sector missing)
3	CRC error	CRC error	
2	Lost data	Lost data	shouldn't happen
1	DRQ	DRQ	always CLR
0	BUSY	BUSY	always CLR

(a) : can be reliably used.

NOTE: All bits are returned in low-true form (i.e., a good sector returns a $FF status).

Figure 6-2. Hardware Status Bits

CHAPTER 7
SPECIAL CHIP FEATURES

This chapter deals strictly with the CHIP itself and illustrates several features of the CHIP which are not fully supported in the ARCHIVER/EDITOR program.

7.1 THE BOOT SECTOR

When the 810 Disk Drive is turned on with the CHIP Modification installed, the head will first align itself on track 0, and then will immediately return to track $27 and read sector $2D0 (if present). The CHIP checks the last two bytes of the sector and compares them to $4A, $25 (or J% in ASCII). If the last two bytes are a $4A and $25 then the program control will be transferred to the sector data for execution. On the ARCHIVER/EDITOR diskette, the boot sector will store a $80 in $195 which will open the drive. It also stores a $02 in $191 which will make the drive shut off one second after it was last accessed. A return is then executed which brings the CHIP's program back to its warm entry.

7.2 MOTOR OFF DELAY

There are two ways to change the motor turn off delay when using the CHIP. The first is to boot a boot sector when you turn on the drive. The other method is to use a built in command which does this automatically. Appendix D is a basic program which first opens the CHIP and then adjusts its motor shutdown delay time.

7.3 LOCKING FORMAT/WRITE/OPEN

The CHIP contains a variable within its memory which allows the opening of the CHIP and of various write type commands. This feature will probably NEVER NEED TO BE USED! However, just in case, location $19D contains the needed information that will TOTALLY lock the CHIP from outside mischief. The modifying of $19D would normally be done in the boot sector, which you would need to write.

7.4 MACHINE LANGUAGE INTERFACE

The CHIP can allow user programs to be transferred to and executed within the data buffer inside of the 810 Disk Drive. This allows for even more flexibility to deal with unforeseen situations, thus the CHIP truly is expandable. For more information on the inner workings of the CHIP, please contact Spartan Software of MN Inc.

7.5 TRACING

The CHIP also supports two types of tracing. One of which keeps track of how many times a particular track is accessed. The other type keeps a listing of the sector numbers read, given some starting sector. These features will be supported by an ARCHIVER 2.0 when released.

CHAPTER 8
USEFUL HINTS

This chapter will deal with tracks and useful things you may do using your ARCHIVER/EDITOR program. This chapter is specifically designed to help the user backup a program that wouldn't work when the defaults were used.

8.1 CYCLIC FORMATS

Consider the following formula:

```
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1 2 3 4 5 6 7 8 9 1 2 3 4 5 6 7 8 9
```

If you write out data using this format you may find that you get a verify error, why? The answer is really quite straight forward. Since all the sectors are doubles, the logic seeking commands will be used, but now how does the logic seeking command locate the sector? It can't because it has no way of distinguishing the first half from the second. The solution to this problem is to turn the logic seeking commands OFF (L-) and the compaction OFF (C-). Also, you should turn the verify off (V-). This will cause each sector to be read in correctly because two sectors will be fetched per revolution and the sectors will automatically be written correctly.

8.2 20 OR MORE SECTORS

The ARCHIVER can only handle reading and writing a maximum of 19 sectors, however, the EDITOR can handle 24. If a diskette does contain more than 20 sectors, the custom formatter must be used and some sectors must be shortened. Notice that 20 full sectors can be written if you set all gaps (except the POST ID CRC) to one (1). However, if more than 20 sectors are being used, you must do some intelligent guessing on which sectors are shortened and go from there. Once you made the format, writing the sectors is easy. The sector sequences must match and the formatting flag should be turned OFF. Also the bad sector flag must be turned to a B- and CRC error bad sector symbols must be created under the sector number (the B command, in the EDITOR of course). Next the sector data must be modified so that the last byte in the bad sectors is the actual number of bytes to be written to the sector. Finally, you write the track and hope it works, otherwise try again. At the time of this writing, NO software company had ever used 20 or more sectors in a format (nor did they have the ability to).

8.3 GARBAGE TRACKS

Occasionally, you may run into tracks that return a read format error. (This has only happened once to my knowledge.) This is because the tracks' are badly garbled and the second pass does not return the same results as the first pass. This will only happen on unformatted tracks, in which case random numbers appear as the sector numbers. To solve this problem, simply switch to a A6- read format mode.

8.4 GETTING RID OF LOUD SECTORS

Many software companies insist on checking missing sectors, thus the loud noises as the program boots. Because most software companies do not check the status after such a read, you may replace their format with a new one that contains the required sectors and the ones that made the noise. When the new format has been created, you must insert bad sectors. The easiest way to do this is to position over the new sector and press the B (first you must get data into that sector). When you have selected all sectors that need to be bad, then write the sectors out, and usually the program will work.

Appendix B ARCHIVER Command Summary

C —		Copy
START	:	start reading/writing
OPTION	:	halt
E —		Enter EDITOR
N —		Number of copies
xy	:	entry (HEX)
O —		Open the CHIP
wxyz,d	:	wxyz is the code, d is the drive
P —		Parameters
←	:	cursor left
→	:	cursor right
RET	:	select parameter
ESC	:	anytime will abort

Appendix C EDITOR Command Summary

A —		ARCHIVER
B —		Bad sector select
D —		Disassemble
	:	scroll up
	:	scroll down
E —		Enter edit mode
↑	:	cursor up one line
↓	:	cursor down one line
←	:	cursor left
→	:	cursor right
DEL	:	delete byte cursor on
INS	:	insert at cursor
CLR	:	fill
H	:	home cursor
RET	:	beginning of line
F —		Formatter
↑	:	cursor up
↓	:	cursor down
→	:	cursor right
←	:	cursor left
DEL	:	delete sector
INS	:	insert sector
CLR	:	delete all sectors
W	:	write format
H —		Hold sector
I —		Insert format

Appendix C EDITOR Command Summary (Continued)

L —		Address change
M —		Enter mapper
xy	:	track number
N —		Renumber current track
xy	:	new number
O —		Open CHIP
wxyz, d	:	CHIP code=wxyz, drive = d
P —		Parameter
→	:	cursor right
←	:	cursor left
RET	:	select parameter
R —		Read tracks
OPTION	:	halt
START	:	begin/continue
W —		Write tracks
OPTION	:	halt
START	:	begin/continue
CLR	—	Delete track
DEL	—	Delete sector
INS	—	Insert sector
ESC	:	return to command mode

Appendix D CHANGING DRIVE MOTOR SHUTDOWN DELAY

```
10 DIM A$(4)
20 ? "WHAT DRIVE DO YOU WANT TO OPEN";
30 INPUT DRIVE:IF DRIVE<1 OR DRIVE>4 THEN 20
40 ? "WHAT IS THAT DRIVE'S CHIP ID CODE";
50 INPUT A$:IF LEN(A$)<4 THEN ? "PLEASE USE 4 DIGITS.":GOTO 40
60 C=0:FOR A=1 TO 4:B=ASC(A$(A,A))-48:B=B-((B>9)*7):C=C*16+B:NEXT A
70 POKE 768,49
80 POKE 769,DRIVE
90 POKE 770,79
100 POKE 771,0
110 POKE 774,15
120 CHI=INT(C/256):CLO=C-CHI*256
130 POKE 778,CLO:POKE 779,CHI
140 RESTORE :FOR A=1 TO 4:READ B:A$(A,A)=CHR$(B):NEXT A
150 X=USR(ADR(A$))
160 IF PEEK(771)<>1 THEN ? "ERROR, TRY AGAIN.":GOTO 20
170 ? "THE CHIP WAS OPENED SUCCESSFULLY."
230 POKE 770,78
240 POKE 771,0
260 ? :? "HOW MANY UNITS OF 1/2 SECONDS DO YOU WANT TO SET THE DRIVE
SHUTDOWN TO";
270 INPUT TIME:IF TIME<1 OR TIME>255 THEN 260
280 POKE 778,TIME
290 POKE 779,0
300 X=USR(ADR(A$))
310 IF PEEK(771)<>1 THEN ? "THE CHIP IS NOT OPEN FOR CHANGE.
PLEASE OPEN IT AND TRY AGAIN.":RUN
320 ? :? "THE DRIVE WAS SUCCESSFULLY MODIFIED."
330 DATA 104,76,89,228
```

FORMAT ERROR	:	After formatting a track, the verify found the track to be bad. Try again, and if it persists, the diskette is likely bad.
READ FORMAT ERROR	:	The CHIP was unsuccessful at getting the sector sequence from the diskette. If you suspect more than 21 sectors, use a **A4** mode., otherwise use a **Ax-** mode.
READ/WRITE ERROR (STD)	:	sector could not be read (or written). This is a standard read/write command and should never happen, unless you have an unreliable drive.
READ/WRITE ERROR (POS)	:	A logic seeking read/write command (sector) failed. Could be a format mismatch problem or an error as in above.
TOO MANY SECTORS	:	More than 25 sectors was encountered on the read format. Try piecing the track together by using **A6-** read mode repeatedly.

INPUT ERROR	:	Invalid entry, try again, or consult appropriate sections regarding the particular function you tried.
VERIFY ERROR	:	The verify pass failed to yield the same results as the data written. Retry the write process.
OPENING ERROR	:	You entered the wrong code or drive of your CHIP when using the O command. Retry the open.
MEMORY FULL	:	No more room to store the data on reads, inserts, etc. Write some of what you have back out to the disk and delete what is not needed.