DISCLAIMER OF WARRANTY

THOMSON SOFTWARE(D.I.P.) does not warrant the contents of this manual to be totally accurate. The information contained in this manual about the TI disk operating system has been thoroughly researched and is believed to be correct. If you notice an error in the manual please bring it to my attention and I will publish a correction.

The user assumes complete responsibility for any decisions made or action taken based on information obtained from this program. THOMSON SOFTWARE(D.I.P.) reserves the right to revise this publication and make changes to the program. When changes or modifications are made, persons on record at THOMSON SOFTWARE(D.I.P.) that have registered copies will be offered the opportunity to acquire these changes.

MAKE SURE THE ALPHA LOCK KEY IS DEPRESSED BEFORE TRYING TO EXECUTE COMMANDS!!!!

1. Loading DISK + AID with Extended Basic

To load DISK+AID from EXTENDED BASIC, plug in the XB command module. Place the disk containing DISK+AID in drive number 1. Select EXTENDED BASIC. The program will automatically load and run. Load time is pretty fast utilizing the speed loader provided. The program cannot be loaded in XB any other way.

2. Loading DISK + AID with Mini-Memory

Plug in MINI-MEMORY and reinitialize memory. Next select the load and run option of MINI-MEMORY. The file name is DISK+AID-O. The program will load and run automatically.

3. Loading DISK + AID with Editor/Assembler

Plug in EDITOR/ASSEMLBER and select the load and run option. The filename to load is DISK+AID-O. The program will load and run automatically.

The first screen you will have is the credit/title screen. Press any key to continue. The next screen you will have will be the WARNING screen that informs you of the dangers of using a single sectoring program.

The MAIN MENU screen displays all of the options available to you for operating DISK+AID. All commands are executed using a single keystroke entry process.

At the top of the screen is the status line. The status line displays the following items:

--Extreme left, last memory area accessed.

--Next is the disk drive number you are working with.

--Current sector number or memory address.

*** WARNING ***

BE SURE YOU HAVE THE SECTOR AND DISK NUMBER YOU WANT TO READ OR WRITE ENTERED BEFORE EXECUTING THE READ OR WRITE SECTOR COMMAND.

--The Caret indicates whether the screen will update after each command. If the caret is present the screen will update. If not the screen will not update.

--The last part will display the last command you executed. Not all commands are displayed. If the cursor is flashing here, you are in the command mode.

O-SELECTING AN OUTPUT DEVICE

This command lets you enter an output device name for storing files on disk or getting printouts.

A-ALTER SECTOR OR MEMORY

This command lets you alter any sector or memory block in either ASCII or HEX. Use the arrow keys to position the cursor where you want it.

B-BACK SECTOR

This command subtracts 1 from the current sector number displayed on the status line.

C-VIEWING CPU MEMORY

This command lets you view a 256 byte block of CPU memory.

D-SCREEN DUMP

This command will take the contents of the screen and dump them to the device you specified enter in the select output device command. The screen dump will work on any valid device except cassette.

E-COMPARING SECTORS

This command will enable you to compare sectors of a disk on one drive or two drives.

F-FORWARD SECTOR NUMBER

This command adds one to the sector number displayed on the status line.

G-VIEWING GROM MEMORY

This command lets you view a 256 byte block of GROM memory.

H-CHANGING MEMORY ADDRESS

This command allows you to manually change the memory address you will be viewing. With a C,G, or V on the status line, you can enter the address next accessed.

I-DISPLAYING SCREEN BUFFER

This command allows you to recall the ASCII or HEX screen that was last read in from memory or disk.

M-MOVING A SECTOR

This command lets you move up to 35 sectors at a time from one place on a disk to another or from one disk to another disk.

N-SECTOR NUMBER

This command will lets you manually enter the sector number you wish. The entry mode works the same as a pocket calculator.

P-PRINTING SECTORS

This command lets you print out one sector or a range of sectors. The printout is sent to the device name you selected with the select output device command. To prematurely stop the printing loop press FCTN 4.

Q-QUITING THE PROGRAM

Press "Q" to quit and go back to the TI title screen.

R-READ SECTOR

This command lets you read the sector pointed to by the disk number and sector number displayed on the status line.

S-SEARCH STRING

This command lets you search for a string up to 40 characters in length in either ASCII or HEX. You can also specify the start and stop sector number. To prematurely stop the search press FCTN 4. With the caret displayed on the status line you can view each sector while its searched. With the caret off the screen will not be updated.

T-TOGGLE BETWEEN ASCII AND HEX

This command lets you toggle the current display between ASCII and HEX.

U-UPDATING SCREEN DISPLAY

This command lets you turn the screen update on or off. See other commands for uses.

V-VIEW VDP MEMORY

This command lets you view a 256 byte block of VDP memory.

FCTN W (~)-WRITING A SECTOR TO A DISK

*** WARNING ***

IF YOU ARE WRITING A SECTOR TO A DISK BE VERY CAREFUL THAT YOU ARE WRITING WHAT YOU WANT WHERE YOU WANT IT. THIS PROGRAM HAS THE POTENTIAL OF DESTROYING DATA AND INFORMATION ON A DISK IF THE USER IS NOT FULLY AWARE OF WHAT THEY ARE DOING !!!!

This command lets you write the a sector of information to disk. Make sure the disk number and sector number on the status line are correct before using this command.

FCTN C (`)-CURRENT CPU MEMORY ADDRESS

This command lets you view the current address you are pointing to in CPU memory.

FCTN G (})-CURRENT GROM MEMORY ADDRESS

This command lets you view the current address you are pointing to in GROM memory.

FCTN Z (\)-CURRENT VDP MEMORY ADDRESS

This command lets you view the current address you are pointing to in VDP memory.

FCTN T ([)-TOGGLE STATUS LINE

This command toggles the middle of the status line between the sector number and the memory address of the area pointed to by the letter on the left side of the status line.

FCTN I (?)-MAIN MENU

This command is your help command. Use it to refresh your memory as to the valid commands and what they do.

FCTN P (")-Mapping Sectors

This command lets you map sector zero or a directory sector. See the section on disk layout for more information.
Any time you map a sector you can also use the screen dump feature to make a permanent record.

1,2,3,4-SELECTING DISK DRIVES

Pressing 1,2,3 or 4 simply changes the disk drive number as indicated on the status line.

FCTN 9 (BACK)-MEMORY BACK PAGE

This command will enable you to page back through memory in 256 byte blocks. The letter on the status line is the memory area you will be back paging thru.

>-NUMBER BASE CONVERSIONS

This command lets you convert 4 digit HEX numbers to 4 digit decimal numbers and vice-versa. The main purpose for this routine is to make converting sector numbers easier.

LAYOUT OF SECTOR 0

** WARNING **

ALTERING SOME INFORMATION COULD CAUSE THE DISK CONTROLLER TO NOT BE ABLE TO ACCESS THE DATA. IT PAYS TO KEEP TRACK OF ANY CHANGES YOU MAKE TO INFORMATION ON THE DISKETTE SO YOU CAN RESTORE IT TO IT'S ORIGINAL CONDITION.

The first thing I'm going to talk about concerning the disk layout will be what is in SECTOR 0. But before I go on, a couple of notes. First of all, whenever you encounter the symbol > this means hexadecimal (hex). Secondly each sector on the disk is divided up into >FF addresses. This corresponds to 256 bytes of storage per sector. In fact as far as the computer is concerned, the disk is nothing more than an extension of memory. The disk controller ROM takes care of dividing up the information to be stored on the disk and writes it to the disk. As the disk controller is doing its job writing to or reading from the disk, it has complete control over the computer. Only after it is done with its job does it return control to your program.

When reading about sector 0 refer to figure 1-1. This is a picture layout of what the sector looks like.

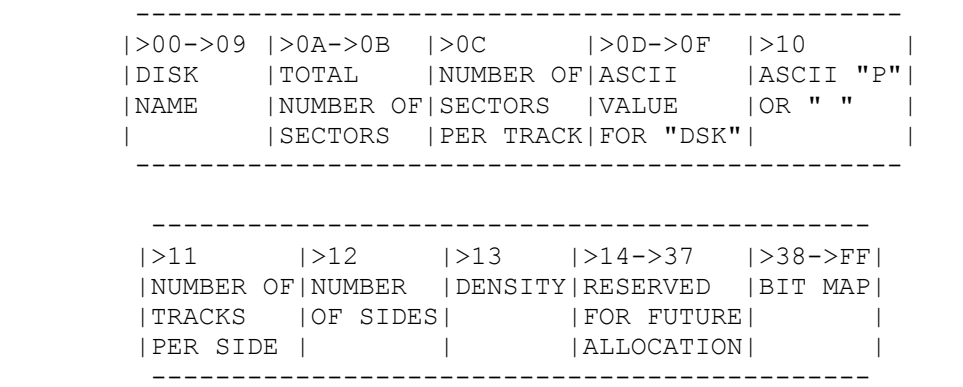

FIGURE 1-1

Each of the blocks in figure 1-1 represents BYTES on the disk in sector 0. I am using hexadecimal numbers because that is the way DISK + AID displays sector information.

Bytes >00->09

These ten bytes contain the name that you assign to the disk when you initialize it with DISK MANAGER command module. The disk name is used by the computer for identification purposes if you ask for the disk by name. Other than that, there is no use for the disk name.

Bytes >0A->0B

These 2 bytes contain the value for the total number of sectors on the disk. If you see >0168 in these 2 bytes that means there are a total of 360 sectors on the disk. If you saw the number >02D0 this would mean there are a total of 720 sectors on the disk. This happens to correspond to the number of sectors on a double sided single density disk. All that is needed to figure out the numbers of sectors is to simply convert the hex number on the disk to a decimal number. The decimal number is the number of sectors on the disk.

Byte >0C

This byte indicates the number of sectors per track. TI-DOS supports only 9 sectors per track.

Bytes >0D->0F

These bytes are the only bytes outside the disk name that I have seen are checked by DISK MANAGER. Byte >0D contains the value >44 which is a "D". Byte >0E contains the value >53 which is an "S". Byte >0F contains the value >4B which is a "K". These 3 bytes are like a flag to DISK MANAGER in telling it whether the disk is initialized or not. If you change these bytes to anything other than "DSK" and try and do anything with DISK MANAGER you will get a "DISK NOT INITIALIZED" message.

Byte >10

This byte contains the value >50 or >20. These values correspond to whether the disk is PROPRIETARY or NONPROPRIETARY. If you see a >50 in this byte it means it is PROPRIETARY. If you see a >20 or any other number for that matter it is NONPROPRIETARY. When you try and copy a disk using DISK MANAGER it checks to see if this byte is set to >50 or not. If not, it will let you copy the disk. If it's set you cannot copy the disk with DISK MANAGER.

There are disk copiers on the market today that will not check to see if a disk is proprietary. The copier will make a complete copy of a proprietary disk. Changing this byte to any value other then >50 or >20 seems to have no effect on the disk controller. The disk can be copied as if it was nonproprietary. It appears the DISK MANAGER only checks to see if >50 is in that location. Any other value and it doesn't seem to care. Before you initialize a new disk, press FCTN X 10 times in a row and then proceed to initialize the disk. It will be proprietary from being copied by DISK MANAGER.

Byte >11

This byte contains the value for the number of tracks per side on a disk. Normally using TI-DOS you can select either >23 or >28 tracks per side. Changing this value seems to have no effect on disk operation. To find out the exact tracks per side in decimal, convert the hex number to decimal.

Byte >12

This byte contains the number of sides on the disk. TI-DOS supports double sided disks so the value in this byte could be >01 or >02 depending on the type of system the disk was initialized on. Changing this value to any value other than the allowed >01 or >02 seems to have no effect with normal disk operation.

Byte >13

This byte contains the density information about the disk. If the value is >01 the disk is single density. If the value is >02 the disk is double density. TI-DOS does not support double density. There are third party systems on the market that will support double density, however, so it is possible that you could run into a disk that has either value in it.

Bytes >14->37

These bytes are reserved for future expansion.

Bytes >38->FF

These 199 bytes are used for the BIT MAP. When you write to a disk the disk controller checks this area to see what sectors are available and what sectors are not.

Each byte in this area represents 8 sectors in use. That breaks down to 1 sector in use for every bit that is set. The first byte is for sectors >000->007. If the first bit is set that means sector 0 is in use. If bit 3 is set that means that sector 3 is in use. The way the byte is read is from right to left. In other words you read it backwards. The table is read from left to right the same way you would read a book. For example in the first byte if these bits are set, 0011 1111, this would mean the sectors >000->005 would be in use. In the second byte if these bits are set, 0110 1100, this would mean that the following sectors would be in use: >00A,>00B,>00D,>00E.

When the disk controller is instructed to write something to the disk, it must first read in this table to determine which sectors can be written to. As it writes to those sectors it will keep track of which ones it actually uses for the particular file. When it closes the file it will write the newly revised bit map back out into sector 0. Be careful if you alter the bit map. If you reset a bit that was supposed to be set and write it back out to sector 0, and you write to that diskette you may write over part of another file as that sector is flagged as being available.

LAYOUT OF SECTOR 1

Sector 1 contains an alphabetized list of all the files on the disk. Every time you add a file the controller reads this list and finds out where the file name would fit in alphabetically. The controller always expects this sector to be in alphabetical order so it does not resort the list each time it accesses it.

Changing the order of the information in this sector can have an effect on the way the disk controller accesses the disk. An

example being, if you have a program on a disk called "LOAD" and it is not listed in alphabetical order with the rest of the programs, say it is further down the list then an "L" should be, Extended Basic will not automatically load the program in like it is supposed to.

Each of the file allocation blocks is 2 bytes long. If a disk had 4 programs on it this is the way Sector 1 would look: >0002000300040005. You will notice that the 2 byte numbers start with a >0002. This is because the first directory sector starts in sector >002. The second directory sector starts in sector >003 and so on.

LAYOUT OF DIRECTORY SECTORS

The next area of the disk I am going to cover is the "DIRECTORY SECTOR". These sectors contain information about a file that is just as important as SECTOR 0 is to the whole disk. The directory sector contains all the information that the controller needs to know about the file. Figure 1-2 is a picture of how a directory sector is laid out.

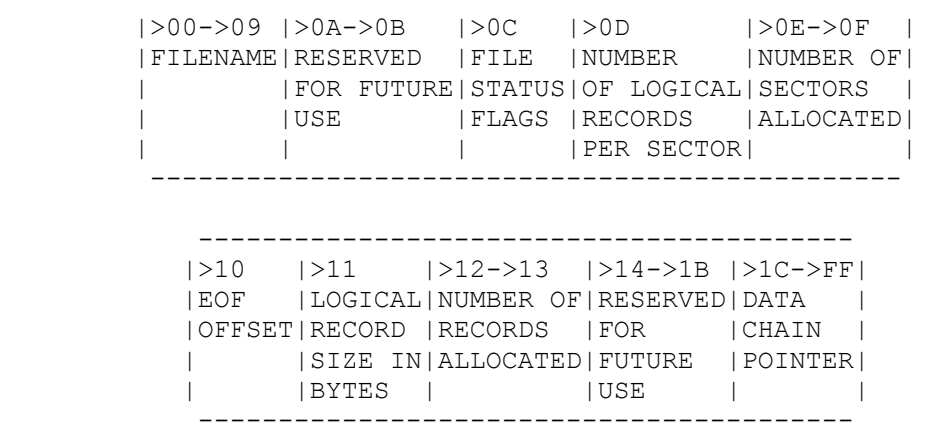

FIGURE 1-2

Each of the blocks in figure 1-2 represent bytes on the disk in every directory sector. I will be explaining each of the blocks in detail, concentrating on the blocks that have the most to do with how the disk controller finds the location of all the information pertaining to any one particular file.

Bytes >00->09

These bytes contain the name you assign to the file. The file name is assigned when the file is created.

Byte >0A->0B

These bytes are reserved for future expansion.

Byte >0C

This byte is broken down into 6 pieces. Each particular piece has a certain meaning as discussed below.

BIT #	DESCRIPTION
0	This bit indicates the type of file: 0 = Data type file 1 = Program type file
1	This bit indicates whether the file is ASCII(display) or BINARY(internal): 0 = ASCII file (display) 1 = BINARY file (internal)
2	Reserved for future data type expansion.
3	This is the PROTECT flag. When you use the DISK MANAGER module and want to modify file protection, this is the flag you are setting or resetting. 0 = Not protected 1 = Protected
4-6	Reserved for future expansion.
7	This bit indicates what type of records the file is written in. This will show the records as being FIXED or VARIABLE. 0 = Fixed length records 1 = Variable length records

Byte >0D

This byte contains the number of logical records per sector for fixed length records only. If you have a FIXED 80 record format, each sector will have a maximum of 3 records with 16 bytes unused. If the file is a program or variable type file this byte will be >00.

Bytes >0E->0F

These 2 bytes contain the number of data sectors for the file.

Byte >10

This byte contains the end of file offset for variable length record files and program files. The value in this byte points to the last valid byte of data in the last sector of the file. The last sector of the file is listed in the sector fragment table. The sector fragment table will be covered in a later section. The sector is flagged as being in use in the bit map in sector 0, even though the sector is not full.

Byte >11

This byte contains the size of the record in bytes. For example a fixed 80 record would have a value of >50 in this byte. If the record is a variable length record, the number indicated here will be the maximum allowable record size. For example let's take a TI-WRITER which is DISPLAY/VARIABLE 80. The value in this byte would be >50. If you change this byte, the disk controller will not know the record size and thus return a error code indicating a file mismatch.

Bytes >12->13

These 2 bytes contain the number of records in a file. If the record length is fixed, the disk controller will calculate how many records will fit in a sector and how many data sectors are allocated for the file. From this it calculates the number of records it has written to the disk.

In the case of variable length records, the disk controller calculates how many records it has actually written out and places that value here. One thing different about this sector is that the 2 bytes are reversed. Normally you would read 2 bytes like >0050 to mean decimal 80. What the disk controller does is take these 2 bytes and reverse them. What you have now is >5000. To the disk controller this means decimal 80.

Bytes >14->1B
These 2 bytes have been reserved for future use.

Bytes >1C->FF
These 233 bytes form what is known as the file data chain pointer block. This is the only place that the controller has to look up the information needed to find all the sectors that make up the file. If you have a lot of patience and determination you may be able to put this block back together again.

The way this table breaks down is: each block of 3 bytes indicates the beginning sector and the offset value for the last sector. The 3 byte blocks break down in an unconventional way that I am going to describe here. First of all you don't read the bytes from right to left. By referring to figure 1-3 while you read you should be able to get a pretty good picture of how the blocks are broken down and the sectors calculated.

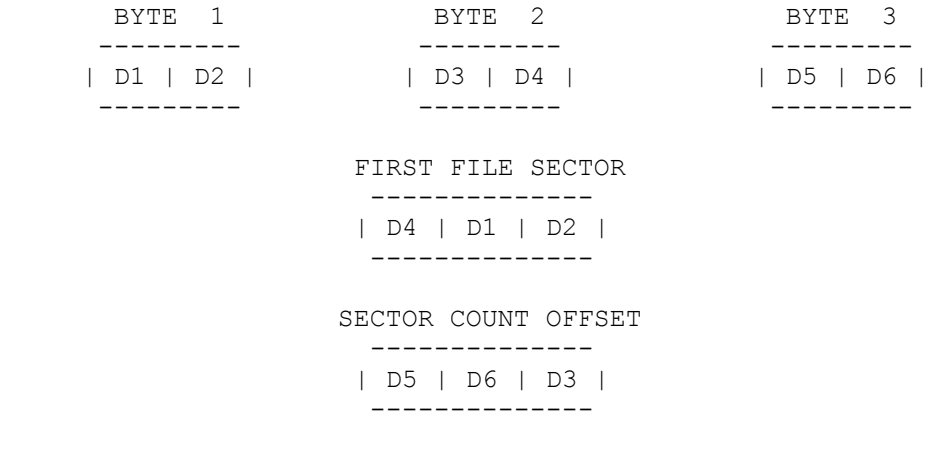

FIGURE 1-3

The first thing that you have to do when figuring out the 3 byte block is to rearrange the hex digits, (D#), as shown above. In our example I will use the 3 byte fragment >E24001. This breaks down to >E2 >40 >01. When the digits are rearranged you will have >0E2 and >014. The first value >0E2 is the first sector in the fragment. The second value >014 is added to the first value to give you an ending value of >0F6. This is the last sector in this fragment.

For files that have more than one cluster the algorithm for figuring each of the clusters beginning and ending sectors is a little more involved. Let's take a file that has 4 clusters in it. For an example I will use a file that has a file data chain pointer table broken down as follows: >6E000073A0007F7001A66002. In the 3 byte sections it would look like this:

1st 3 bytes = >6E >00 >00
2nd 3 bytes = >73 >A0 >00
3rd 3 bytes = >7F >70 >01
4th 3 bytes = >A6 >60 >02

For this table you would figure the first cluster the same way as you did before. In this case the first fragment will start at >06E with an offset of >000. So, the first and last sectors in the fragment would be >06E.

The second set of 3 bytes breaks down to a starting sector value of >073 and an offset of >00A. This is where all similarities between calculating the first cluster and the remaining clusters end.

The next step is to subtract the offset value in the first cluster from the offset value in the second cluster.

```
Second cluster offset =  >00A
First cluster offset  = ->000
                        -----
                         >00A
```

Now subtract 1 from the answer, >00A, which leaves you with >009. Add this value to the cluster starting sector of >073. You will end up with >07C. This is the last sector in the second cluster.

When the digits are rearranged in the 3rd cluster you end up with a starting sector of >07F. The calculated offset will be >017. Subtract the value of the 2nd cluster offset from the value of the 3rd sector offset:

```
Third cluster offset  =  >017
Second cluster offset = ->00A
                        -----
                         >00D
```

Subtract 1 from the >00D which leaves you with >00C. Add this value to >07F and you end up with >08C. The >08C is the last sector in the 3rd cluster of the table.

The 4th cluster breaks down to a start sector of >0A6 and an offset of >026. When you plug the values in for the offset calculation:

```
Forth cluster offset =  >026
Third cluster offset = ->017
                       -----
                        >00F
```

you end up with an offset of >00F as seen above. Subtract 1 from >00F to give you >00E. Add >00E to >0A6 to get the value of the last sector in the cluster, >0B4.

As you can see, calculating the file data chain pointer table can be a real challenge. For one cluster, it's not too bad but for multiple clusters it can get pretty confusing keeping everything straight.

If you would like the full 46 page manual which includes a 4 pages section on recovering lost data and a much more detailed overview of using DISK+AID, send $5.00 to cover printing and mailing costs to:

THOMSON SOFTWARE(D.I.P.)
3507 MURL
MUSKEGON, MI 49442