C O M M O D O R E

DISK REFERENCE MANUAL

FOR

D9090 D9060 8250 8050 4040 2031

NOTICE

The information in this manual has been reviewed and is believed to be entirely reliable. However, Commodore assumes no responsibility for any inaccuracies. This is a **preliminary** manual which is provided for information purposes only and is subject to change without notice. It is being provided now, in preliminary form, so as not to delay introduction of new disk-storage products. A more complete and comprehensive version of this manual is being prepared and will be available soon.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

INTRODUCTION

Read the Table of Contents to become acquainted with the broad scope of material covered in this manual, which has been designed to assist you in using the computer as an aid to the learning process. Once the disk drive is properly interfaced to a Commodore Computer, the worth and utility of the system, is measured in direct relationship to how well you learn to use the hardware and software.

This manual presents material specific to the Commodore 5 1/4 - inch hard Winchester Disk Drives, the 8250 dual drive, double sided disk unit, the 8050 and 4040 single sided dual disk units and the 2031 single disk unit. Commands and procedures will, for the most part, work on all models. The exceptions will be described in the appropriate sections of this manual.

Users who have attained some degree of programming skills may desire to begin with the advanced subjects such as advanced file handling, while others may be content with following the manual's format. In either case, you are provided with essential information in a logical sequence. Follow the examples, attempt the step-by-step procedures, and learn by doing.

GENERAL INFORMATION

With the purchase of the Commodore Disk system, you have greatly enhanced the computing power of your Commodore computer system. All Commodore disk units are "intelligent" peripherals, therefore requiring no computer memory for operation. This means that you have just as much computer memory available, whether the disk drive is attached or not. To get the most out of the system you should study both the computer user's guide and this manual.

DESCRIPTION OF THE D9060 & D9090

The two models of hard Disk Drives described in this manual are single-drive storage devices. The primary components consist of read/write controls, drive motor electronics, a drive mechanism, two or three platters with recording surfaces on both sides, four or six read/write heads, and track positioning mechanisms. The disk drives conform to PET-IEEE interface requirements. An IEEE interface connector is located on the back of the drive. Near the lower edge of the rear panel is a power ON/OFF switch. There is also a "slow blow" fuse, and an AC power cord.

DESCRIPTION OF THE 8250

The model 8250 dual floppy disk unit uses a 100 Track per Inch (TPI) two headed drive with a formatted capacity of 1,066,496 bytes (characters) per drive. Each 8250 diskette has 154 tracks, 77 on each side, and is read/write compatible with the model 8050 disk drive. The 8250 uses Micropolis Tandon drives.

DESCRIPTION OF THE 8050

The model 8050 dual floppy disk unit uses a 100 Track per Inch (TPI) single headed drive with a formatted capacity of 533,248 bytes per drive. Each 8050 diskette has 77 tracks, and is read/write compatible with the model 8250 disk drive. This compatibility is limited to one side of the diskette. The 8050 uses either Micropolis or Tandon drives.

DESCRIPTION OF THE 4040

The model 4040 dual floppy disk unit uses 48 track per inch (TPI) single headed drives with formatted capacities of 174,848 bytes (characters) per drive. Each 4040 diskette has 35 tracks. The 4040 is neither read nor write compatible with the model 8050 or the 8250 disk drives. Diskettes created on 4040 drives are read/write compatible with the model 2031 and the VIC-1540 disk units.

DESCRIPTION OF THE 2031

The model 2031 is a low-cost single drive disk unit. The 2031 uses a 35 track (48 TPI) single headed drive with a formatted capacity of 174,848 bytes. Diskettes created on 2031 drives are read/write compatible with the model 4040 and VIC-1540 disk units.

PREPARING TO USE THE DISK UNIT

Before starting to use the disk drive, make sure it is in good working condition. This includes properly connecting it to the computer, giving it power-on and initial checkout tests, and finally running performance tests using the TEST/DEMO diskette (except hard disk).

Commodore disk units described in this manual are operationally compatible with any model PET or CBM computer equipped with BASIC 3.0 or BASIC 4.0. VIC-20 and Commodore-64 computers equipped with the appropriate PET-IEEE adapter cartridge can also use these disk units.

UNPACKING THE DISK UNIT

Before unpacking the disk drive, inspect the shipping carton for signs of external damage. If the carton is damaged, caution should be excercised when inspecting its contents. The contents and all packing material should be removed from the carton. NO packing materials should be discarded until all the contents are located. The carton should contain:

1. Model D9060, D9090, 8250, 8050, or 4040 Disk
2. User Manual
3. TEST/DEMO diskette (except for D9060 or D9090 hard disk units)

If any item is missing, your Commodore dealer should be notified.

CONNECTING THE DISK UNIT TO THE COMPUTER

One of two connector cables are required to interface the disk drive to the computer. These cables can be supplied by your Commodore dealer.

PET-to-IEEE cable: Part # 320101
This cable shoud be used if the disk drive is to be connected directly to the computer.

IEEE-to-IEEE cable: Part # 905080
This cable shoud be used if the disk drive is to be connected ('daisy-chained') to another peripheral device such as the Commodore Model 4022 or any other suitably interfaced printer.

NOTE: The disk drive should be the first peripheral attached to the computer if other devices are to be 'daisy-chained'.

Procedure for connecting the disk drive to the computer:

STEP 1: Power to the computer and all peripherials should be turned OFF.

STEP 2: The disk drive should be located as close as possible to the computer.

STEP 3: The PET-to-IEEE cable connects between the IEEE-488 interface on the computer and the disk drive. If additional IEEE devices are to be connected, the IEEE-to-IEEE cable(s) must be used.

STEP 4: The disk unit power cable should be connected to an AC outlet at this time, but with its power switch turned OFF.

PERFORMING THE POWER-ON TEST

Procedure for power-on checkout:

STEP 1: Power should now be applied to the computer to verify that it is working properly. The following message will be displayed:

```
*** Commodore Basic ***
31743 Bytes Free  (Depends on Memory Size)
ready
```

STEP 2: Power should now be applied to the disk drive. All indicator lights (LEDS) on the front panel should flash twice. The two-color power/error LED should stay green, indicating power ON.

If the drive lights remain on or all lights flash continuously or if the power/error LED stays red for more than five seconds, turn the power OFF. Wait a moment and try again. If these conditions are repeated, all other devices should be removed from the IEEE bus. This will assure that a possible problem related to another device does not affect the disk unit. If the problem persists, your Commodore dealer should be contacted.

NOTE: After applying power to D9060 or D9090 Hard Disk units, WAIT ONE FULL MINUTE before attempting to use any disk command. This time is required to allow rotational speed of the disk to stabilize. Any commands issued before this time will cause a DRIVE NOT READY error message to occur, and the drive will not respond to further commands until the INITIALIZE command is used.

HANDLING DISKETTES

Unless you have one of the hard disk drives, your disk unit will have either Micropolis, Tandon, or Shugart drive mechanisms. Suggested procedures for inserting diskettes into the drive(s) differ from model to model. However, as a general precaution (for all models) to help insure proper seating of diskettes, be sure that the diskette is reasonably well centered in its casing before inserting it into the drive.

1. **Micropolis - 8250:** The Micropolis drives used in the 8250 include switches to detect diskette insertion; however, closing the door does not start the drive motor. For proper seating of diskettes the 'flip down' door should be 'teased' once or twice before final closing.

2. **Micropolis - 8050:** The 8050 Micropolis drive starts the drive motor as the diskette is locked in position. The gate should not be 'teased' shut, which may cause a mis-seating of the diskette. Simply insert the disk and press the gate latch down firmly, but gently, without hesitation.

3. **Tandon - 8050:** Two versions of the Tandon drive are supplied. The versions are visually identical; however, the later version has a switch to start the drive motor to aid in seating disks. To properly seat a disk in earlier Tandon drives, the 'flip-down' door should be 'teased' once or twice before final closing. For later version drives, the motor startup will seat diskettes without the teasing.

4. **Shugart - 4040 / 2031:** Diskettes will seat properly in the 4040/2031 Shugart drives by just closing the 'flip-down' door without teasing. These drives do not have diskette insertion detect or automatic motor start. Thus, the INITIALIZE command (Chapter 2) should always be used after changing diskettes, before any other command is used.

Floppy diskettes are fragile but they can be a long-lasting and very reliable data storage medium when handled properly. Always treat your diskettes gently; never force them into the disk drive. Keep them in their paper sleeves when not in use - in a case designed to hold them. Keep diskettes away from magnetic fields, as found near electric motors or power transformers. Never set heavy objects, such as cups, bottles or books on top of diskettes. Be prepared for the **inevitable** unforseen accident: MAKE FREQUENT BACKUP COPIES of your data - and keep the copies in a safe place.

Any 'soft-sectored' single-density or double-density certified diskette will work reasonably well with Commodore floppy disk units. However, for the 8050 and 8250 disk units, double-density diskettes are recommended. Diskette hub rings also help long-term reliability of any diskette.

COMMODORE DISK SYSTEM SPECIFICATIONS

MODEL	D9090	D9060	8250	8050	4040	2031
Drives per Unit	1	1	2	2	2	1
Heads per Drive	6	4	2	1	1	1
Formatted Storage Capacity per Unit	7.47 Mb	4.98 Mb	2.12 Mb	1.05 Mb	340 Kb	170 Kb
Max Sequential File	7.41 Mb	4.94 Mb	1.05 Mb	521 Kb	168 Kb	168 Kb
Max Relative File	7.35 Mb	4.90 Mb	1.04 Mb	183 Kb	167 Kb	167 Kb
Disk System Buffer RAM (Bytes)	4 Kb	4 Kb	4 Kb	4 Kb	4 Kb	2 Kb
DISK FORMATS						
Cylinders (Tracks)	153	153	77	77	35	35
Sectors/Cylinder	128	192	-	-	-	-
Sectors per track	32	32	23-29	23-29	17-21	17-21
Bytes per sector	256	256	256	256	256	256
Blocks Free (Unit)	29162	19442	8266	4104	1328	664
TRANSFER RATES (Bytes/Sec)						
Internal to Unit	5 Mb	5 Mb	40 Kb	40 Kb	40 Kb	40 Kb
IEEE-488 Bus	1.2 Kb	1.2 Kb	1.2 Kb	1.2 Kb	1.2 Kb	1.2 Kb
ACCESS TIMES (Milli-seconds)						
Track-to-track	3	3	5	*	30	30
Average Track	153	153	125	**	360	360
Head settling time	15	15	-	-	-	-
Average Latency	8.34	8.34	100	100	100	100
RPM	3600	3600	300	300	300	300

* Track-to-track: Micropolis 8050 = 30 ms. Tandon 8050 = 5 ms.
** Average Track: Micropolis 8050 = 750 ms. Tandon 8050 = 125 ms.

PHYSICAL DIMENSIONS						
Height (in.)	5.75	5.75	7.0	7.0	7.0	5.5
Width (in.)	8.25	8.25	15.0	15.0	15.0	8.0
Depth (in.)	15.25	15.25	13.75	13.75	13.75	14.25
Weight (lbs.)	21	21	28	28	28	20
ELECTRICAL						
Power (watts)	200	200	60	50	50	40
Voltage (all Models)		110 - 120 VAC, 60 Hz				

CHAPTER 2

LEARNING HOW TO USE THE DISK DRIVE

b	byte
d	destination
dn	device #
dr	drive #
fn	file name
ft	file type (REL, SEQ, USR, PRG)
lfn	logical file #
mode	READ or WRITE
n	new
o	old
r	record #
R	read
rl	record length
s	source
sa	secondary address
vn	variable name (eg. A, X1$)
W	write
xx	two character diskette identifier

NOTE: The description of disk commands and all examples of usage that follow are given in BASIC 4.0 syntax. Disk command syntax used with BASIC 3.0 (or earlier) are shown in Chapter 7. It should also be noted that since the hard disks are single-drive units, use of two-drive commands is not valid for those devices. The **Commodore BASIC 4.0 Reference Manual** may be consulted for additional information on disk commands.

PREREQUISITES

Commodore disk units add to and enhance the computing power of your system with added storage and file handling capability. They are controlled directly with:

- o BASIC commands entered via the keyboard
- o BASIC statements within programs, and
- o Special disk commands.

In this manual you will learn how to apply disk commands and statements. Functions and format of disk commands, which permit the user to perform disk-related tasks, are described here. For BASIC 4.0 users, those BASIC commands which correspond to each disk maintenance command are also discussed.

This chapter will advance toward an understanding of those BASIC commands used for data handling. The disk commands should be practiced along with the examples, and the illustrations should be followed. The understanding of the more advanced disk programming techniques will depend to a large degree upon how well the fundamentals have been mastered.

To facilitate understanding and mastery of Commodore BASIC, some computer terms are stressed in this Chapter: Files, Disk Operating System (DOS) and Block Availability Map (BAM). Although these are conventional terms, they will be briefly discussed as they relate to Commodore disk usage.

FILES

For the novice computer user, an explanatory word about files is in order. A file is an organized set of information which is stored on some form of media, be it tape, diskette, or even something as mundane as a filing cabinet.

With regard to computers, however, any information entity which is stored external to computer memory is a file. This requires that files be given names for identification and that some means of locating them be provided. The Disk Operating System (DOS) takes care of most of these details.

A file can be a set of program instructions (a program file) or a set of data (such as a list of names or an inventory of auto parts) which a program can access to store and retrieve information.

THE DISK OPERATING SYSTEM (DOS)

The DOS is responsible for managing information exchange between the disk controller and the computer. The DOS performs many functions which are transparent to the user but which are vital to the operation of the system. For example, the DOS monitors the input/output (I/O) of the disk so that channels are properly assigned and that no lengthy waits for an open channel occur. In addition to monitoring of disk I/O, the DOS also uses the channel structure to search the directory and to delete and copy files.

THE BLOCK AVAILABILITY MAP (BAM)

The BAM is a disk memory representation of available and allocated (used) space on a disk. Formatting a disk creates the BAM which is then loaded into DOS memory upon initialization. The BAM is stored on disk in various locations, depending upon the model of disk unit.

When the system stores data on a disk, the BAM will be referenced by the Disk Operating System (DOS) to determine whether space is available. For Sequential or Program files the DOS checks for space before each block of the file is written. If a free block is found, the BAM is updated to account for the space used and the data block is written to the disk. If no free blocks are available an error message will be generated.

As changes occur to the BAM in DOS memory, the BAM on disk will be updated periodically to reflect these changes. Updates to the BAM occur when a program is DSAVEd or when DCLOSE is performed on a Relative or Sequential data file. One block of the BAM is loaded into DOS memory at a time. When updated, this block is written back to the disk and the other block loaded into memory. This interchange of information between the two BAMs, one in DOS memory and the other on disk, enables the system to maintain a record of free and allocated space on the disk.

COMMUNICATING WITH DOS

Input/output programming can become complex when data is transfered to or from cassette drives, printers, disk drives and external devices other than the screen or keyboard. For these more complex operations a 'channel' must be opened between the program and the selected device by using the CBM BASIC OPEN statement. After performing required operations the channel must be CLOSEd.

Each device attached to the computer has its own unique physical device number (8 thru 15 for disk units) to which it responds when being accessed by the computer. The device number is also used as a parameter when OPENing a channel, to identify the physical unit to be accessed.

CBM disk units are preset at the factory to respond to physical device number 8; however, their device number may be changed (see Memory-Write command Chapter 4) by means of DOS commands. The DOS recognizes an optional device number specification for all Disk Maintenance commands and some Data File commands (Chapter 3). If device number is not specified DOS will assume unit number 8.

Channel numbers (which can range from 0 thru 255) do not have permanent assignments, but are assigned arbitrarily by the programmer. Thus, channel numbers are refered to as 'logical file' numbers. The 'logical file number' relates OPEN, CLOSE, INPUT, GET and PRINT statements with each other and associates them with the physical device being accessed.

In addition to a logical file number and a device number, Commodore disk units also respond to several 'secondary addresses'. Secondary addresses are best visualized as 'commands' from the computer telling the disk unit what operation it is to perform.

Address 0 is used (with DLOAD) to read a program file into computer memory. Address 1 is used (with DSAVE) to write programs from memory into a diskette program file. Addresses 2 - 14 are used to access data files. Address 15 is a special 'command/status' address used to perform many of the special disk operations described in this manual and to retrieve status information about disk operations.

FILE NAME PATTERN MATCHING

Pattern matching of file names is available on all Commodore disk units. Pattern matching uses question marks (?) and asterisks (*) to perform an operation on several files with similar names using a single command. The asterisk is used at the end of a string of characters to indicate that the rest of the name is insignificant and is to be ignored in the search for matching file names. For example "FIL*" could refer to files named:

```
      FIL
or    FILE1
or    FILEDATA
or    FILLER
or    Any other file name starting with the letters FIL.
```

The question mark may be used anywhere within the string of characters to indicate that only the character in that particular position should be disregarded. For example "?????.SCR" could refer to files named:

```
      TSTER.SRC
or    DIAGN.SRC
or    PROGR.SRC
      but not  SRC.FILES
```

Both the characters and the position of the characters are significant.

The question mark and asterisk may be combined in many useful ways. However, the asterisk should always appear as the last character in any pattern, whether or not question marks are used. For example the pattern "J*?????" does not make sense because the question marks are in an area which is insignificant (because of the asterisk).

The pattern "P???FIL*" will access files with the names:

```
      PET FILE
or    PRG-FILE-32
or    POKEFILES$$
or    Any other file starting with 'P' and having 'FIL' in positions 5-7.
```

DLOAD "*" will load the first program file in the disk directory. DOPEN with pattern matching may be used to open an existing file, in which case the first file encountered which fits the pattern will be opened. DOPEN must NOT be used with pattern matching when creating a new file.

The SCRATCH command with pattern matching should be used carefully, since multiple files will be scratched. Never use RENAME, DSAVE, or COPY with pattern matching, since an error condition will result.

DISK MAINTENANCE COMMANDS

The following disk commands permit file manipulation and disk maintenance. The disk-level commands provide for formatting disks to ready them for use, displaying a directory of file names contained on disks, initializing a drive to assure that DOS 'knows' which blocks on a disk are used and unused and for re-building a valid BAM in the event of software failures.

The file-level commands provide for copying files from one drive or disk unit to another, appending one file to another, changing the names of files and for removing unwanted files from the disk directory.

	FUNCTION	COMMAND NAME
Disk Level	Format a disk	HEADER
	Read Block Availability Map (BAM) into DOS buffer	INITIALIZE
	Read disk directory	DIRECTORY
	Reconstruct Block Availability Map	COLLECT
File Level	Copy files	COPY
	Copy with append	CONCAT
	Rename a file	RENAME
	Erase a file	SCRATCH

NOTE: The drive number reference in all DOS commands has been maintained in the examples which follow to be compatible with dual drive disk units. If using the 2031 single disk or the D9060/D9090 hard disk drives, all references to drive number must be a zero (0). Any reference to drive 1 will result in an error condition.

VARIABLE COMMAND PARAMETERS

Each disk command has associated with it one or more optional parameters which may be used to specify file names, drive numbers, device numbers, etc. When needed, command parameters may appear in either of two forms. Parameters may be stated explicitly, such as: DOPEN#1,"Inventory File",D1 or BASIC variable names enclosed in parentheses may be used, such as: DOPEN#1,(A$),(DN). The two DOPEN commands above would produce the same results.

When entering disk commands from the keyboard in direct mode, parameters must be stated explicitly. When used in programs, parameters may either be explicit or variable, and both forms may be used in the same command.

COMMAND ABBREVIATIONS

Whether entered in direct mode or used in a program, DOS commands may appear either with their full spelling or in abbreviated form. Commands are abbreviated by entering enough characters of the command name to uniquely distinguish it from any other DOS command or BASIC keyword. All but the last character of the abbreviation are keyed unshifted and the last character shifted.

For example, 'catalog' and 'cA' are identical to the DOS, as are 'print#1' and 'pR1'. Abbreviation of commands does not reduce memory usage in programs, but is supported as a convenience for user of the system. When a program containing command abbreviations is listed, the commands will appear in fully spelled form.

HEADER

A previously unused diskette must first be formatted in the soft-sector format recognized by your disk unit by using the HEADER command. This process writes track/sector addresses on the disk, writes binary zeros to all blocks, and creates the BAM, Directory Header and the Directory.

Since formatting destroys any data previously stored on a disk, the HEADER command has a built-in safety feature that first queries: ARE YOU SURE? Typing a "y" in response permits the disk to be HEADERed. Any other response aborts the operation without writing on the disk.

Format:

```
HEADER "diskname" [,Ddr] [onu#] [,Ixx]
```

Example:

```
HEADER "Inventory",d0 onu9,i02
```

The above statement will format the disk in drive # 0 on unit # 9, giving it the name "Inventory" and disk identification number (ID) of "02". Hard disk units (D9060 and D9090) must also be HEADERed before using them to store data. If the drive number is omitted DOS will default to drive # 0.

The DOS provides for 'quick formatting' of previously used disks. If the disk ID number is omitted, the DOS will create a new Directory Header (the disk name may be changed) and write an 'empty' BAM and Directory to the disk, but without writing zeros to all data blocks.

INITIALIZE

Whenever a diskette is inserted into a drive, for any reason, that drive MUST be initialized to ensure that the BAM data in DOS memory is the proper data for the diskette currently in the drive. Failure to properly initialize the drive may cause a DISK ID MISMATCH error, or loss of data.

Format:

```
PRINT#lfn,"Idr"
```

Example:

```
OPEN 15,8,15
PRINT#15,"I0"
CLOSE 15
```

Initializes drive # 0 of the disk unit. If the drive number is omitted then both drives will be initialized, on dual disk units.

The 4040 and 2031 disk units check diskette ID each time the disk is addressed to find whether initialization is needed. If a new diskette ID is detected the drive is initialized without need for operator action. If the ID is the SAME as a previous diskette the change of diskettes WILL NOT be detected and data will be lost if the drive is not initialized.

Since the 8050 and 8250 disk units feature automatic detection of diskette removal/insertion, these units will self-initialize either when the door is closed (8050 Micropolis) or when the drive is first addressed (8250 and 8050 Tandon).

DIRECTORY / CATALOG

This command will display a listing of the file names stored on a disk. The contents of computer memory are not disturbed. The directory display includes the following information:

- o Disk Name, Disk ID and DOS Version
- o File name and File Type
- o File size (number of blocks used)
- o Number of available (free) blocks on the disk

Format:

```
DIRECTORY [Ddr] [onu#]    or    CATALOG [Ddr] [onu#]
```

Example:

```
DIRECTORY D1    or    CATALOG D1
```

These will cause the directory for drive 1 to be displayed. Omission of the drive number will cause the directory of BOTH drives to be displayed in succession on dual drive units. Unit numbers other than 8 may be used. To list a disk directory on a Commodore printer, the following command sequence may be used.

```
OPEN1,4          Opens a channel to device 4 (printer).
CMD1             Switches the screen output to device 4.
DIRECTORY        Prints the directory.
PRINT#1          Returns output to the screen.
CLOSE1           Closes the channel to the printer.
```

By changing the OPEN statement above to the format: DOPEN#1,"name" a sequential data file would be created on disk, called 'name', containing the contents of the disk directory. This provides a convenient means of accessing the directory as input data to a program.

COLLECT

The COLLECT command traces through each block of data contained in all files on the disk. If this trace is successful, a new BAM is generated in the disk memory and written to the disk. Any blocks which have been allocated but are not associated with a file name, as in the case of direct access files (User file type) will be freed for other use.

In addition to reconstructing the BAM, COLLECT deletes files from the directory that were never properly closed. If a READ error is encountered during a COLLECT, the operation aborts and leaves the disk in its previous state. If a COLLECT error does occur, the drive must be initialized before proceeding.

COLLECT accomplishes the following:

- Recreate a Block Availability Map (BAM) according to valid data on the disk.
- Delete files from the directory which were never properly closed (DOPENed but never DCLOSEed).

Format:

```
COLLECT [Ddr] [onu#]
```

Example:

```
COLLECT D0
```

This will reconstruct the BAM, delete any unclosed files on drive 0 and free any blocks reserved via the Block-Allocate command (see Chapter 4). Omitting the drive number will cause both drives to be COLLECTed on dual drive units. The device number specification (onu#) is optional.

COPY

This command creates an identical copy of a file, either on a different drive (or device number) or, with a different file name, on the same disk.

Format:

```
COPY [Dsdr] [onu#] ,"sfn" TO [Dddr] [onu#] ,"dfn"
```

Example:

```
COPY D0,"names" TO Dlonu9,"friends"
```

This will copy a file called "names" on drive 0 unit # 8 to drive 1 on unit # 9. The name of the new file is "friends". The COPY command may be used with pattern matching to copy several files at a time. If the file name already exists on the destination disk, an error condition results and the copy is not done. Omission of source drive number causes a search of both drives for the file. Destination drive number defaults to 0.

CONCAT

Concatenation of files amounts to copying one SEQUENTIAL file and appending its data to the end of another SEQUENTIAL file. The source file remains unchanged and its contents are appended to the destination file which then contains all records from both files.

Format:

```
CONCAT [Dsdr] [onu#],"sfn" TO [Dddr] [onu#],"dfn"
```

Example:

```
CONCAT D0,"names" TO D1,"friends"
```

This will add the data in a file called "names" on drive 0 to a file called "friends" on drive 1. The file "names" on drive 0 will remain unchanged and "friends" will contain all data from both files. The CONCAT command may only be used on SEQuential files. Omission of source drive number causes a search of both drives. Destination drive defaults to 0.

RENAME

The RENAME command changes the name of an existing file. The file name to be changed must be properly closed. The new file name must not currently exist on that drive or the error message FILE EXISTS will be generated and the file name will not be changed.

Format:

```
RENAME [Ddr] [onu#],"o fn" TO "n fn"
```

Example:

```
RENAME D1, "clients" TO "patients"
```

This will change the file named "clients" on drive 1 to "patients". Drive number will default to 0 is not present. Pattern matching may not be used to rename files. Both drives will be searched if drive number is omitted.

SCRATCH

The SCRATCH command removes unwanted files from the directory. The file is not 'erased', instead the directory entry for the file is marked as SCRATCHed by setting its File Type (Directory Block Formats, Chapter 6) to

zero. The blocks occupied by the file are marked as available in the BAM. A built-in DOS safety feature queries the user: ARE YOU SURE? Typing a "y" in response permits the file to be SCRATCHed. Any other response aborts the operation.

Format:

SCRATCH [Ddr] [onu#],"fn"

Example:

SCRATCH d1,"data-x"

Are you sure ? y

This will cause the file named "data-x" on drive # 1 to be scratched and the blocks occupied by that file freed for other use. Pattern matching may be used to SCRATCH several files at a time. If drive number is omitted, both drives will be searched for files.

CHAPTER 3

BASIC COMMANDS

For FILE HANDLING

DATA FILE COMMANDS

The BASIC commands described in this chapter, allow the user to open, communicate with, and transfer data to and from files on the disk unit. It should be noted that these are not program (PRG) files, but rather data files that are either RELative or SEQuential. A SEQuential file is one in which information is stored and retrieved in sequence, one record after the other, i.e. the file must be searched from the beginning up until the desired information is found. A RELative file is one which allows direct access to any piece of information.

All characters shown as upper-case in the following formats are essential for the proper execution of a command and must appear exactly as shown. These commands are entered via the keyboard (using unshifted characters only) or they may be used in BASIC programs. Characters shown in lower case or within quotes represent parameters supplied by the programmer.

DSAVE"fn"	PRINT#lfn,vn
DLOAD"fn"	INPUT#lfn,vn
DOPENlfn,"fn"	GET#lfn,vn
DCLOSE#lfn	RECORD#lfn,R,B

DSAVE

This command transfers the program currently in computer memory to a file on the disk. The DOS will flag the file as a program (PRG) file type.

Format:

DSAVE [Ddr] [onu#],"fn"

Example:

DSAVE D1,"payroll acct"

This command will save a file named "payroll acct" to drive 1 on the disk. The file name may be any name up to 16 characters, including blanks. If drive number is omitted the program will be stored on drive # 0. An optional device number (onu#) may be specified.

DLOAD

The DLOAD command transfers PRG files from the specified disk to the computer's memory. The user must specify the program name. A successful DLOAD closes all open files. Therefore the user must give a new DOPEN command in order to continue communicating with the disk drive command and error channels. (DOPEN will be discussed later in this chapter).

Format:

DLOAD [Ddr] [onu#],"fn"

Example:

```
DLOAD "customers", D0
```

A program file named "customers" will be loaded from drive 0 into the computer's memory. The drive number will default to 0 if not specified.

QUICK LOAD FEATURE: For computers with BASIC 4.0, pressing the SHIFT and the RUN/STOP keys simultaneously causes the first program file in the directory of drive # 0, unit # 8 to be loaded into memory and executed.

DOPEN

This command sets up a correspondence between a logical file number and a Relative or Sequential file on disk. It also reserves buffer space within the disk unit for operations on the file being opened. The format is slightly different for each file type.

Format (SEQuential files):

```
OPEN#lfn,"filename",Ddr,W (or R)
```

Example:

```
DOPEN#1,"account",D0,W
```

This will open a SEQuential file called "account" on drive 0 for writing to disk. The file name "account" must not currently exist on the disk or the error message FILE EXISTS will be displayed. If a non-existant file is opened for reading, the error message FILE NOT FOUND is displayed. If the W is omitted or an R is used, DOS will open the file for reading.

Format (RELative files):

```
DOPEN#lfn,"filename",Ddr,Lrl
```

Example:

```
DOPEN#1,"account",D0,L128
```

This will open a RELative file called "account" on drive 0 with a record length of 128 characters. If the file name "account" does not currently exist, the file will be created. If the named file does exist, it will be opened for both reading and writing.

Using the DOPEN command on an already open file will cause an error which automatically closes the file. To recover, simply DOPEN the file again.

DCLOSE

This command closes files opened with the DOPEN command, updates the BAM to reflect the last block(s) allocated to the file and updates the file's directory entry to show the number of blocks occupied by the file.

Format:

DCLOSE# lfn

Example :

DCLOSE#25

This statement will close logical file number 25. If the logical file number is omitted all currently open disk files will be closed.

NOTE: It is good practice to always close a file after working with it. A maximum of ten open files in the computer and five in the disk drive are permitted, therefore it is prudent to make a habit of closing files as soon as possible. This way you will always have the maximum number of files available for use.

PRINT#

The PRINT# statement is used to transmit data to a previously DOPENed SEQuential or RELative file. Using BASIC 4.0, any file opened with logical file number greater than 127 will automatically result in a line feed character CHR$(10) being transmitted to the device when a carriage return character CHR$(13) is sent. Logical file numbers less than 128 will suppress the line feed character. Note that earlier versions of BASIC do not support this feature.

Format:

PRINT#lfn,variable name

Example:

PRINT#8,Y$

This will cause the value of Y$ to be written to logical file number 8.

Several variables may be written to the disk at the same time.

Example:

PRINT#lfn,A$;B$;C$

This will result in variables A$, B$, and C$ being concatenated into a single data string stored on disk. Note that the variable names in the PRINT statement are separated by semicolons.

Example:

PRINT#lfn,ACHR(13)BCHR(13)C$

This will result in writing the variables A$,B$, and C$ separated by carriage returns. They may then be retrieved as separate variables using the INPUT or GET statements.

INPUT#

The INPUT# command is used to transfer information from the disk drive into computer memory. INPUT# is valid only when referencing a logical file that has been DOPENed for input.

Format:

```
INPUT#lfn,variable name
```

Example:

```
INPUT#5,A$
```

This will read data from logical file number 5 into variable name A$. The input is terminated by encountering (whichever occurs first): a maximum of 80 characters, a carriage return CHR$(13), a comma CHR$(44) or a semi-colon CHR$(59).

Example:

```
INPUT#5,A$,B$,C$
```

In this example, the data strings must have been separated by some delimiter character at the time they were written to the disk in order to be retrieved separately. No single string may contain more than 80 characters if it is to be INPUT. For strings longer than 80 characters, the GET# command must be used.

GET#

The GET# command is used to tranfer individual bytes of information from an IEEE device such as the disk unit into computer memory. GET# is valid only when referencing a file that has been DOPENed for reading.

Format:

```
GET#lfn,variable name
```

Example:

```
GET#15,A
```

In this example a single byte of numeric data will be retrieved from logical file number 15 into the variable named A.

When using the GET statement to retrieve string data, if the data read is a binary zero (or null character) the variable used to hold it will have a length attribute of zero. For proper handling of later operations, such as comparisons, the null must be converted to its CHR$ value as shown.

Example:

```
GET#7,B$:IF B$="" THEN B$=CHR$(0)
```

The GET# statement may be used to transfer several bytes of data, which is useful for retrieving strings which have been written to the disk in a format unacceptable for the INPUT command (longer than 80 characters).

Example:

```
AA$=""
FOR I=1 TO 254
GET#12,A$
AA$=AA$+A$
NEXT
```

This program segment would result in a string of length 254 being transferred from logical file number 12 to computer memory and stored in the variable AA$.

RECORD#

The RECORD# command is used prior to a PRINT#, INPUT#, or GET# in order to position the file pointer to the desired record (and byte) of a RELative file. For example, if the file pointer is set beyond the last record and PRINT# is used, the appropriate number of records are generated to expand the file to the desired size.

Format:

```
RECORD#lfn,r,b
```

Example:

```
RECORD#15,12,8
```

This will position the RELative record file pointer to byte number 8, of the 12th record, in logical file number 15. The byte number (8) is optional and if omitted the file pointer will be positioned to the first data character of the record.

Example:

```
RECORD#1,25
INPUT#1,A$
```

Inputs the next record as a string and assigns it to variable A$. A detailed discussion of the RECORD command for relative file manipulation is found in Chapter 5.

CHAPTER 4

ADVANCED

DISK PROGRAMMING

This chapter provides detailed information about DOS structure and disk utility commands. The utility commands provide the programmer with low-level functions that may be used for special applications such as special disk handling routines and random access techniques.

Overview of DOS Versions

DOS 2.1 works with the 4040 dual disk unit. Model 2040 disk units can be upgraded to DOS 2.1 by replacement of ROM chips. Reliability of the recording format of DOS 2.1 was improved over DOS 1.0 by removing one block from tracks 18 thru 24. As a result the directory holds 144 file entries and 664 blocks for user data.

The Relative Record file structure was added to DOS 2.1 to provide for random access to files. The Block Read/Write commands of DOS 1.0 are supported, but the corresponding 'U1' and 'U2' utility commands should be used for upward compatibility with future CBM disk products.

In general, software which does not depend upon physical device attributes should be upward compatible for all versions of DOS. Programs using the Block Read/Write commands are very vulnerable to DOS changes.

DOS 2.5 is used in all 8050 dual disk units. All of the features of DOS 2.1 are included in DOS 2.5 and adapted for additional capacity. DOS 2.5 also includes enhancements such as disk insertion detect and expanded error recovery techniques. The directory provides 224 file entries and 2052 blocks are available for user data.

DOS 2.6 is used in the 2031 single disk unit. DOS 2.6 is a functional equivalent to DOS 2.1 (used in the 4040) and is fully compatible with DOS 2.1 with one exception. Since the 2031 is a single-drive unit, dual drive commands will not operate on the 2031.

DOS 2.7 is used in the 8250 double-sided dual disk unit. DOS 2.5 disk commands and the 8050 disk unit are upward compatible with DOS 2.7 and the 8250. With certain restrictions diskettes created on either disk unit are read/write compatible. One important feature of the 8250 is the Expanded Relative File capability of DOS 2.7 which allows these files to occupy an entire 8250 diskette, providing a capacity of over 1 million bytes.

DOS 3.0 is used in the D9060 and D9090 hard disk units. Features of DOS 3.0 include a dynamically expandable directory allowing an unlimited number of file entries, replacement-mapping of bad sectors, and a self-locating BAM.

General Operation of DOS

The DOS file interface controller is responsible for managing all data transfers between the IEEE bus and the disk controller. Most disk I/O is performed on a pipelined basis, resulting in faster response to requested operations.

The file system is organized by channels which are opened with the BASIC DOPEN statement. When the DOPEN statement is executed, the DOS assigns a workspace to each channel and allocates one to three disk I/O buffer areas. If either the workspace or the buffer is not available, a NO CHANNEL error is generated. The DOS also uses the channel structure to search the directory, and to delete and copy files.

The common memory between the disk controller and the file interface controller is used for 256-byte buffer areas. Three of the sixteen buffers are used by the DOS for the Block Availability Map (BAM), variable space, command channel I/O, and the disk controller's job queue.

The job queue is the vital link between the two controllers. Jobs are initiated on the file side by providing the disk controller with sector header and type of operation information. The disk controller seeks the optimum job and attempts execution. An error condition is then returned in place of the job command. If the job is unsuccessful, the file side re-enters the job a given number of times, depending upon the operation, before generating an error message.

The secondary address given in the OPEN statement is used by DOS as the channel number. The number the user assigns to a channel is only a reference number that is used to access the work areas, and is not related to the DOS ordering of channels.

The DLOAD and DSAVE statements transmit secondary addresses of 0 and 1, respectively. The DOS automatically interprets these secondary addresses as DLOAD and DSAVE functions. Unless these functions are desired when opening files, avoid secondary addresses of 0 and 1. The remaining numbers, 2 through 14, may be used as secondary addresses to open up to five channels for data.

DISK UTILITY COMMAND SET

The disk utility command set consits of the following commands:

Commands	Abbreviations	General Format
BLOCK-READ	B-R	"B-R:"ch;dr;t;s
BLOCK-WRITE	B-W	"B-W:"ch;dr;t;s
BLOCK-EXECUTE	B-E	"B-E:"ch;dr;t;s
BUFFER-POINTER	B-P	"B-P:"ch;p
BLOCK-ALLOCATE	B-A	"B-A:"dr;t;s
BLOCK-FREE	B-F	"B-F:"dr;t;s
Memory-Write	M-W	"M-W"adl/adh/nc/data
Memory-Read	M-R	"M-R"adl/adh
Memory-Execute	M-E	"M-E"adl/adh
USER	U	"Ux:ch;dr;t;s

ch=the channel number in DOS: identical to the secondary address in the associated OPEN statement.

dr=the drive number: 0 (or 1 for floppy dual drives)

t=the track number: 1 thru 154 (depending on Model #)

s=sector number: 0 thru 112 (depending on Model #)

p=the pointer position for the buffer pointer.

adl=the low byte of the address.

adh=the high byte of the address.

nc=the number of characters: 1 through 34.

data=the actual data in hexidecimal. This is transmitted by using the CHR$ function, i.e. CHR$(17) would send the decimal equivalent of hexadecimal 11.

x=the index to the User Table.

parms=the parameters associated with the U command (optional).

These commands may be abbreviated to the first character of each of the key words. Only abbreviations are accepted for the MEMORY Read, Write and Execute commands. DOS searches for parameters associated with each command starting at a colon(:), or in the fourth character position if a colon is not present. The following example shows four ways that the same BLOCK-READ command may be given.

Examples:

```
"BLOCK-READ:"2,1,4,0
"B-R"2,1,4,0
"B-R"2;1;4;0
"B-READ:"A;B;C;D
```

Parameters following the key words within quotation marks may be separated by any combination of <cursor-right>, SPACE or Comma characters. If using variable names to pass command parameters, only the command string should be enclosed in quotes as shown in the general format examples above.

BLOCK READ

This disk utility command provides direct access to any block on the disk. Used in conjunction with other block commands, a random access file system may be created through BASIC. This command positions the DOS file pointer to the first character or "0-position" of the block. When a character in this position is read with GET# or INPUT#, an End-or-Identify (EOI) is sent. This terminates an INPUT# and sets the Status Word (ST) to 64 in the computer.

Format:

"B-R:"ch;dr;t;s

Example:

"B-R:5;0;18;0

Reads the block from drive 0, track 18, sector 0, into channel 5 buffer area.

After using BLOCK-READ to transfer the data to the buffer, the data may be transferred to memory by INPUT# or GET# from the logical file opened to that disk channel (i.e., using that secondary address). The U1 command described under USER is similar to the BLOCK-READ command.

BLOCK-WRITE

When this command is initiated, the current buffer pointer is used as the last character pointer and is placed in the 0-position of the new buffer. The buffer is then written to the indicated block on the disk and the buffer pointer is left in postion 1.

Format:

"B-W:"ch;dr;t;s

Example:

"B-W:"7;0;35;10

Writes channel 7 buffer to the block on drive 0, track 35, sector 10. The U2 command described under USER is similar to the BLOCK-WRITE command.

BLOCK-EXECUTE

This command allows part of the DOS or user designed routines to reside on disk, be loaded into disk drive memory, and be executed. The File Interface Controller begins execution of the contents after the block is read into the specified buffer. Execution must be terminated with a "Return From Subroutine" (RTS) instruction. Future system extensions or user-created functions may implement this feature.

Format:

"B-E"ch;dr;t;s

Example:

"B-E:"6;0;1;10

Reads a block from drive 0, trackl, sector 10, head 0 into channel 6 buffer and executes its contents beginning at position 0 in the buffer.

BUFFER-POINTER

This command changes the pointer associated with the given channel to a new value. This is useful when accessing particular fields of a record within a block or, if the block is divided into records, individual records may be set for transmitting or receiving data.

Format:

```
"B-P:"ch,p
```

Example:

```
"B-P:"2;1
```

Sets channnel 2 buffer pointer to the beginning of the data area in the direct access buffer.

BLOCK-ALLOCATE

This command requests that the DOS flag the block on the specified drive track and sector as being "in use". If successful, the appropriate Block Availability Map (BAM) is updated in DOS memory to reflect the block as allocated (used). In future operations, the DOS skips over the allocated block when saving programs or writing files. The updated BAM is written to disk upon closing an output file or closing the command channel.

If the block requested has been previously allocated, the error channel indicates the next available block (increasing track and sector numbers) with a NO BLOCK error. If no blocks are available, greater in number than the one which was requested, zeroes are shown as the track and sector parameters when the NO BLOCK error is returned.

Format:

```
"B-A:"dr;t;s
```

Example:

```
"B-A:"0;10;0
```

Requests that block (sector) 0 of track 10 Head 0 be flagged as allocated on the disk. Always check the error channel when using this command to prevent an allocated block from being overwritten. If the block is allocated, the error message will also indicate the next available block.

Example:

```
INPUT#15,EN,EM$,ET,ES
```

Reads the next available track and sector, respectively, into ET and ES. Assumes that lfn=15 was previously OPENed to the disk command channel.

```
EN=Error Number        EM$=Error Message
```

MEMORY COMMANDS

All three MEMORY commands are byte-oriented so that the user may utilize machine language programs. BASIC statements may be used to access data via the MEMORY commands by using the CHR$ function. The system accepts only M-R, M-W, and M-E; neither verbose spelling or the use of the colon (:) is permitted. The INITIALIZE command must be sent (once only) to a drive before issuing a sequence of MEMORY commands to that drive.

MEMORY-WRITE

This command provides direct access to the DOS memory. Special routines may be down-loaded to the disk drive via this command and then executed using the MEMORY-EXECUTE command or one of the USER commands. Up to 34 bytes may be deposited with each use of the M-W command. The hexadecimal value of the DOS memory address must be specified low-byte first and must be converted to decimal for use with the CHR$ function.

Format:

```
"M-W"adl/adh/nc/data
```

Example:

```
"M-W"CHR$(0)CHR$(18)CHR$(4)CHR$(32)CHR$(0)CHR$(17)CHR$(96)
```

Writes four bytes to buffer 2 ($1200 or decimal 4608). Another use for the M-W command is to temporarily change the physical device number of a disk unit. All disk units are set to device number 8 at the factory. When two or more disk units are attached to the computer, the device number of each unit must be made unique or none will operate correctly. The following program fragment will change the device number of 4040, 8050 or 8250 floppy disk units and D9060 or D9090 hard disk units.

Example:

```
OPEN15,odn,15
"M-W"CHR$(12)CHR$(0)CHR$(2)CHR$(ndn+32)CHR$(ndn+64)
CLOSE15
```

To change the device number of 2031 disk units use this M-W statement:

Example:

```
"M-W"CHR$(119)CHR$(0)CHR$(2)CHR$(ndn+32)CHR$(ndn+64)
```

The general procedure to change device number is:

1. power-up the first disk unit only.
2. run the above program.
3. then power-up the next disk unit.

The device number will remain at the new value until changed again by the M-W command or a (U: or UJ) command is issued or the unit is powered down.

MEMORY-READ

The byte pointed to by the DOS memory address in the command string may be accessed with this command. Variables from the DOS or the contents of the buffers may also be read with this command. The M-R command changes the contents of the error channel since it is used for transmitting data to the computer. The next GET# from the error channel (secondary address 15) transmits the byte. An INPUT# should not be executed from the error channel after a MEMORY-READ command until a DOS command other than one of the MEMORY commands is executed.

Format:

```
"M-R"adl/adh
```

Example:

```
"M-R"CHR$(128);CHR$(0)
GET#15,A$
```

This will access and reads the byte located at ($0080 hexadecimal or decimal 128).

MEMORY-EXECUTE

Subroutines in the DOS memory may be executed with this command. To return to the DOS, terminate the subroutine with an RTS instruction.

Format:

```
"M-E"adl/adh
```

Example:

```
"M-E"CHR$(128)CHR$(49)
```

Requests execution of the code beginning at $3180 hexadecimal.

USER COMMANDS

These commands provides a link to 6502 machine code according to a jump table pointed to by the special USER pointer. The second character in this command is used as an index to the table. The ASCII character 0 through 9 or letters A through O may be used. Zero sets the USER pointer to a standard jump table that contains links to special routines.

The special USER commands U1 (or UA) and U2 (or UB) can be used to replace the BLOCK-READ and the BLOCK-WRITE commands on all DOS versions. Because of errors in DOS 2.1 the B-R and B-W commands do not operate correctly in 4040 disk units. Thus B-R and B-W must be replaced by U1 and U2 when programming for the 4040.

The U1 command forces the character count (buffer pointer) to 255 and reads an entire block into memory. This allows complete access to all

bytes in the block, including the track/sector link pointer.

Format:

"U1:"ch;dr;t;s

Example:

"U1:"5;0;18;0

The block at track # 18, Sector # 0 on drive # 0 is read into buffer channel number 5. The data may then be accessed via the M-W and GET# commands.

U2 writes a buffer to a block on the disk without changing the contents of position 0 as B-W does. This is useful when a block is to be read in (with B-R) and updated (B-P to the field and PRINT#), then written back to disk with U2.

Format:

"U2:"ch;dr;t;s

Example:

"U2:"5;0;18;0

Writes the data in channel buffer # 5 to drive # 0, track 18, sector 0.

STANDARD USER JUMP TABLE

STANDARD DESIGNATION	ALTERNATE DESIGNATION	FUNCTION
U1	UA	BLOCK-READ replacement
U2	UB	BLOCK-WRITE replacement
U3	UC	jump to $1300
U4	UD	jump to $1303
U5	UE	jump to $1306
U6	UF	jump to $1309
U7	UG	jump to $130C
U8	UH	jump to $130F
U9	UI	jump to $10f0 (NMI)
U:	UJ	Power-up Vector

The U3 thru U9 commands are user-defined. The locations jumped to are located in the buffer areas of disk unit RAM. User-written DOS routines may be coded to reside there and may be downloaded from the computer using the M-W command or read from disk using the B-W or U1 commands. Location $10F0 is the location of the NMI interrupt handler. The U: or UJ commands cause the disk unit to perform its power-up sequence and resets device number. The drive(s) must be initialized before issuing further commands.

CHAPTER 5

ADVANCED FILE HANDLING

RELATIVE FILES

The preceding chapters explained how to manipulate files on the disk, and described the format of commands used to create and manipulate sequential files. In this chapter, these skills will be utilized in discussing direct access file handling using Relative Record files.

RELATIVE FILES

Direct access (or RELative files) is a method that allows the programmer to position a pointer to any record on the disk relative to the beginning of that file. Compare this method to the standard procedure of having to search each track and sector for the desired information and it becomes apparent that such a relative handling method would result in a great reduction in the amount of time required to find and fetch a specific record stored on disk.

The three main components of a relative file are the **super side sector** (DOS 2.7 and DOS 3.0 only), **the side sector** chain of blocks and the **data block chain**. All are linked together through forward pointers similar to those used in a sequential file.

The super side sector points to the first side sector in a group of side sectors. Each side sector points to other side sectors in the same group and points to a data block chain. Record sizes, while fixed in length, may range from 1 to 254 bytes. The number of records is limited (under DOS 2.1 and DOS 2.5) to that which can be contained in 720 data blocks, as each side sector can contain a maximum of 120 data block pointers. The number of records under DOS 2.7 and DOS 3.0 is limited to the capacity of the disk but for practical purposes should not exceed 65,535.

The side sectors do not contain record information, but do contain pointers to the data blocks. The record size dictates where the pointer is positioned when a record number is referenced because the record size is used in an algorithm to compute where the pointer is positioned when a record number is given through the RECORD command.

The side sector also contains a table of pointers to all of the other side sectors within the file. In order to move from one side sector to another, the pointer is referenced through the appropriate DOS command, and the corresponding side sector is read into memory. Using information contained in the referenced side sector, the data block pointer can be located and used to read in the actual data block containing the record.

The relative file data block pointers in the side sectors allow the DOS to move from one record to another within two disk read commands - a considerable savings in the amount of time required to find a desired data block when compared to sequential methods.

Each **side sector** contains pointers for 1 to 120 data blocks. There are six side sectors for each relative file under DOS 2.1 (4040) and DOS 2.5 (8050). This provides a total file capacity of 182,880 bytes (120 pointers/side sector * six side sectors * 254 bytes/data block). The **super side sector** of DOS 2.7 (8250) and DOS 3.0 (Hard Disks) has the capacity to point to 127 groups of side sectors, giving a total capacity of 23,225,760 bytes per file (182,880 bytes * 127 groups of side sectors).

Spanning of data blocks is a key feature of relative files which aids in reducing the number of disk read/write operations required to find and retrieve data. Before explaining how this feature of DOS improves time utilization efficiency, we need to examine how I/O channels are utilized by relative files:

When a channel is opened to a previously existing file, the DOS will position to the first record provided that the given parameters match properly. The record length variable is not necessary on the DOPEN statement if the file already exists. The DOS checks the record size (if specified) against the record size that appears in the directory entry for an existing file. If these do not match, then an error message will be generated.

Relative files require three memory buffers from the system, whereas sequential files only require two. Since there are twelve buffers in the system and two of these are used in directory searches and internal functions, only three relative files can be open at once. The highest number of buffers that can be used is ten, which limits the total number of channels which can be open at any one time.

If a record was found to be on the boundary between two data blocks, that is, starting in one data block and finishing in another, then the DOS would read the first segment as well as any following records in the second data block. In practice, the records of most relative data files will span across data blocks. The only exceptions are record size 1, 2, 127, and 254. These divide evenly into the 254 size of the data block and spanning is unnecessary.

This method of spanning has the advantage of requiring no system memory overhead aside from that required for the side sector blocks in the relative files. When a record is written via the PRINT# statement, the data block is not immediately written to disk. It is only written out when the DOS moves beyond the particular data block in which that record resides. This can occur through successive printing to sequential records, or when positioning to another record outside of that particular block.

Because of the spanning feature, it is imperative that multiple channels NOT be open to a single relative file at the same time if any channel will be writing to the file. An update may be made in one channel's particular memory buffer area, but the change may not be made on disk until the DOS moves beyond that particular data block. DOS places no restriction on this, and when the file is open for READ only, it may be advantageous to have multiple channels open to a single relative file.

The DOS terminates printing to a record by detecting the EOI signal which is generated with each PRINT# statement. If the PRINT# statement goes over the maximum record size an error message will be generated. Any data overflow will be truncated to fit the number of character specified by the record size and the DOS will position to the next record in sequence.

If the print statement contains less characters than the actual record size, the remaining positions within that record will be filled with nulls or binary zeros. Consequently, when positioning to a record for input the EOI signal is generated from the DOS to the computer when the last non-null byte is transmitted. Should the programmer desire to store binary information, a record terminator such as carriage return must be used and the record size increased by one character to accommodate the terminator.

CREATING A RELATIVE FILE

When a relative file is opened for the first time, the file should be initialized by the programmer to allow for faster subsequent access, and to assure that the DOS reserves sufficient space on the disk for the future data. A relative file may be initialized by first opening the file, setting the file pointer to the last (highest) record number to be contained in the file, printing to that record, and then closing the file.

Example:

```
DOPEN#1,"FILE1",D0,L50
RECORD#1,100
PRINT#1, CHR$(255)
DCLOSE#1
```

In the preceding example the DOPEN creates a file on drive # 0 with the name FILE1 and a record length of 50.

The RECORD# statement positions the file pointer to record number 100 which does not yet exist. The error message 50 RECORD NOT PRESENT will occur at this point, but should be interpreted as a warning rather than an error condition. This message is normally expected to occur as a warning when a new record is accessed for the first time and indicates that no INPUT or GET operation should be attempted.

The PRINT# statement causes record number 100 to be written. During this write operation, the DOS detects that records 1 thru 99 do not already exist, and automatically initializes them by placing CHR$(255) in the first character of each record. During this process, all necessary side sectors and data block pointers are also created.

While the DOS is generating new data blocks for relative files, the requested record number is compared to the number of data blocks left on the disk. If the resulting number of data blocks is greater than the number available on the disk, then error 52 FILE TOO LARGE is generated.

The DCLOSE statement closes the file and causes space to be allocated in the BAM and updates the block count in the file's directory entry.

After the file has been initialized, data may be written to the file. Initialization of a file in this manner need be performed only once when the file is originally created.

EXPANDING A RELATIVE FILE

To expand an existing file, the same procedure as for creation is used, with the record number changed to reflect the greater number of records.

When DOPEN is used on an existing relative file, the record length parameter is optional. If present, it must match the length set at the time the file was created or error 50 RECORD NOT PRESENT will result.

When a file is expanded in this manner, required side sectors are also

created. Side sectors are transparent to the user since they are automatically generated and accessed by the DOS.

ACCESSING A RELATIVE FILE

In order to make the relative file system practical, the user must be able to access the file for reading and writing of data. Both of these operations are simplified by relative files and both may use the RECORD command for positioning to the desired record before the operation.

To write data to or read from a predetermined record in a file, the RECORD# statement is used to set the DOS file pointer to the desired record. The record number parameter may be a constant or a BASIC variable name enclosed in parentheses as shown.

Example:

```
DOPEN#1,"FILE1",D0
RECORD#1,25
or:  RECORD#1,(RN)          Where RN has the value 25
PRINT#1,"Philadelphia"
DCLOSE#1
```

The resulting record would appear as follows:

```
        1         2         3         4         5
12345678901234567890123456789012345678901234567890
_____________________________________________________
Philadelphia*
_____________________________________________________
```

Where * represents a carriage return CHR$(13).

The following program illustrates an optional feature of the RECORD# statement which permits access to individual bytes within a record for writing or reading.

```
DOPEN#1,"FILE1",D0
RECORD#1,25              (Sets file pointer to record 25)
PRINT#1,"Philadelphia"
RECORD#1,25,20           (Sets character pointer to position # 20)
PRINT#1,"Penna."
RECORD#1,25,30           (Sets character pointer to position # 30)
PRINT#1,"19204"
DCLOSE#1
```

The following illustration is a representation of the contents of record number 25 after the above example is executed:

```
        1         2         3         4         5
12345678901234567890123456789012345678901234567890
_____________________________________________________
Philadelphia*     Penna.*   19204*
_____________________________________________________
```

NOTE: It is important that the fields be written in sequence, since writing to a byte at the beginning of the record destroys the rest of the record in DOS memory. This means that while it is possible to position and write first to byte 1 and then to byte 20, it is NOT possible to first write byte 20 and then byte 1.

Since the carriage return is recognized as a terminator by the BASIC INPUT# statement, the data in the preceeding example may be retrieved by the following sequence:

Example:

```
DOPEN#1,"FILE1",D0
RECORD#1,25
INPUT#1,A$          (Reads "Philadelphia" into A$ variable).
RECORD#1,25,20
INPUT#1,B$          (Reads "Penna." into B$ variable).
RECORD#1,25,30
INPUT#1,C$          (Reads "19204" into C$ variable).
DCLOSE#1
```

The RECORD# command may be omitted if the file is to be accessed sequentially, which saves time during program execution. An example of this occurs when writing a large data base to the disk file. Assume that the program has already dimensioned variable D$ as an array which contains 100 elements. These elements are to be written to the disk in records number 1 thru 100 of file FILE1. This could be accomplished with the following program segment:

Example:

```
DOPEN#1,"FILE1",D0
FOR I=1 TO 100
PRINT#1,D$(I)
NEXT I
DCLOSE#1
```

Since the record pointer is automatically set to record 1 when the file is opened, record 1 is the first record written. If no RECORD command is executed the DOS automatically positions to the next record after each PRINT#. Therefore, the contents of D$ array elements will be written to records 1 thru 100 of the file.

USING 8050 DISKETTES IN 8250 DRIVES

Although the 8050 and 8250 disk units are read/write compatible, the first access to an 8050 diskette inserted into an 8250 drive (or use of the Initialize command) will cause an error 66 ILLEGAL TRACK OR SECTOR message. The message occurs because of the different BAM contents of the two disk systems and may be ignored. The error will occur only once and all further disk commands will operate correctly unless the diskette is moved to another drive.

For ease of use, data on 8050 diskettes should be transfered to 8250 formatted diskettes using the COPY command. The BACKUP command will not work for this.

The 8050 disk unit is upward compatible (read/write) to the 8250 with some exceptions. The 8050 disk unit cannot access the reverse (top) side of an 8250 formatted diskette. Relative files created on an 8250 disk unit cannot be accessed by an 8050 unless the Expanded Relative File feature of the 8250 was disabled before creating the file and unless the file resides entirely on the (bottom) diskette surface that the 8050 can access.

MANAGING RELATIVE FILES ON THE 8250

Relative files on 8050 disk units are limited to a size of 182,880 bytes. On 8250 disk units with DOS 2.7 this limit no longer applies and relative files may use the entire capacity of an 8250 diskette. The 8250 will power-up with the Expanded Relative File feature enabled. To read/write 8050 formatted relative files, this feature must be disabled as follows:

Example:

```
OPEN 15,8,15
PRINT#15,"M-W"chr$(164)chr$(67)chr$(1)chr$(255)
CLOSE 15
```

This disables access to expanded relative files until the 8250 is powered down or reset by a (U: or UJ) USER command or until the Expanded Relative File feature is re-enabled as follows:

Example:

```
OPEN 15,8,15
PRINT#15,"M-W"chr$(164)chr$(67)chr$(1)chr$(0)
CLOSE 15
```

Existing relative files in 8050 format can be converted to the 8250 Expanded Relative File format by means of a program named "EXPAND.REL" which is included on the TEST/DEMO diskette supplied with 8250 disk units. To convert 8050 relative files to 8250 format DLOAD and RUN this program (you must use an 8250 disk unit). A series of instructions will be displayed on the screen. The expanded relative files output by this program cannot be accessed by an 8050 disk unit.

CHAPTER 6

DISK STORAGE FORMATS

This chapter provides the details of disk storage fromats of the 4040, 8050, and 8250 floppy disk units and the D9090 and D9060 hard disk units. For each type of disk the tables which follow show: Block Distribution by track, locations and formats of the Block Allocation Map, the Directory Header, the Directory, and the formats of Program, Sequential, and Relative files.

BLOCK DISTRIBUTION BY TRACK

Disk Unit	Track Nr.	Nr. Blocks
2031	1 - 17	21
	18 - 24	19
	25 - 30	18
	31 - 35	17
4040	1 - 17	21
	18 - 24	19
	25 - 30	18
	31 - 35	17
8050	1 - 39	29
	40 - 53	27
	64 - 64	25
	65 - 77	23
8250	1 - 39	29
	40 - 53	27
	54 - 64	25
	65 - 77	23
	78 - 116	29
	117 - 130	27
	131 - 141	25
	142 - 154	23

D9060/D9090: 153 tracks per recording surface (4 on D9060, and 6 on D9090) with 32 sectors per track.

BAM (Block Allocation Map) FORMATS

2031 BAM Format - Track 18 Sector 00

Byte	Data	Definition
0-1	18-00	Track-Sector of first directory block
2	65	ASCII 'a' identifies DOS 2.6 format
3	00	Reserved for future DOS use
4-143		Bit map of available blocks, tracks 1-35

4040 BAM Format - Track 18 Sector 00

Byte	Data	Definition
0-1	18-00	Track-Sector of first directory block
2	65	ASCII 'a' identifies DOS 2.1 format
3	00	Reserved for future DOS use
4-143		Bit map of available blocks, tracks 1-35

8050 BAM (First Block) Format - Track 38 Sector 00

Byte	Data	Definition
0-1	38-03	Track-Sector of second BAM block
2	67	ASCII 'c' identifies DOS 2.5 format
3	00	Reserved for future DOS use
4	01	Lowest track # mapped in this BAM block
5	51	Highest track # (+1) mapped in this BAM block
6		Nr. of unused blocks on track # 1
7-10		Bit map of available blocks on track # 1
11-255		Bit map of available blocks, tracks 2-50

8050 - Second BAM Block Format - Track 38 Sector 03

Byte	Data	Definition
0-1	39-01	Track-Sector of first directory block
2	67	ASCII 'c' identifies DOS 2.5 format
3	00	Reserved for future DOS use
4	51	Lowest track # mapped in 2nd BAM block
5	78	Highest track # (+1) mapped in 2nd BAM block
6		Nr. of blocks unused on track # 51
7-10		Bit map of available blocks on track # 51
11-140		Bit map of available blocks, tracks 52-77

8250 BAM (First Block) Format - Track 38 Sector 00

Byte	Data	Definition
0-1	38-03	Track-Sector of second BAM block
2	67	ASCII 'c' identifies DOS 2.7 format
3	00	Reserved for future DOS use
4	01	Lowest track # mapped in 1st BAM block
5	51	Highest track # (+1) mapped in 1st BAM block
6		Nr. of unused blocks on track # 1
7-10		Bit map of available blocks on track # 1
11-255		Bit map of available blocks, tracks 2-50

8250 - Second BAM Block Format - Track 38 Sector 03

Byte	Data	Definition
0-1	38-06	Track-Sector of third BAM block
2	67	ASCII 'c' identifies DOS 2.7 format
3	00	Reserved for future DOS use
4	51	Lowest track # mapped in 2nd BAM block
5	101	Highest track # (+1) mapped in 2nd BAM block
6		Nr. of blocks unused on track # 51
7-10		Bit map of available blocks on track # 51
11-255		Bit map of available blocks, tracks 52-100

8250 Third BAM Block Format - Track 38 Sector 06

Byte	Data	Definition
0-1	38-09	Track-Sector of fourth BAM block
2	67	ASCII 'c' identifies DOS 2.7 format
3	00	Reserved for future DOS use
4	101	Lowest track # mapped in 3rd BAM block
5	151	Highest track # (+1) mapped in 3rd BAM block
6		Nr. of unused blocks on track # 101
7-10		Bit map of available blocks on track # 101
11-255		Bit map of available blocks, tracks 102-150

8250 - Fourth BAM Block Format - Track 38 Sector 09

Byte	Data	Definition
0-1	39-01	Track-Sector of first directory block
2	67	ASCII 'c' identifies DOS 2.7 format
3	00	Reserved for future DOS use
4	151	Lowest track # mapped in 4th BAM block
5	155	Highest track # (+1) mapped in 4th BAM block
6		Nr. of blocks unused on track # 151
7-10		Bit map of available blocks on track # 151
11-255		Bit map of available blocks, tracks 152-154

D9060/D9090 BAM Block Format - Track 1 Sector 0 (Normal Location)

Byte	Data	Definition
0-1		Track-Sector pointer to next BAM block (hexadecimal $ffff = last BAM block)
2-3		Track-Sector pointer to previous BAM block (hexadecimal $ffff = first BAM block)
4		Lowest track # mapped in this BAM block
5		Highest track # (+1) mapped in this BAM block
6		Nr. of blocks unused on this track
7-10		Bit map of available blocks on this track
11-255		Bit map of next 49 tracks

Structure of BAM Entries for one Track - All DOS Versions

Each track has five bytes allocated to map it. A map bit=1 means the block is available; bit=0 means the block has been used. Blocks are mapped by bytes, the high order bit of each mapping the lowest numbered block of each group.

Byte	Definition
1	Current number of available blocks for this track
2	Bit map blocks 0 - 7. Bit 7 = block 0, bit 0 = block7
3	Bit map blocks 8 - 15. Bit 7 = block 8, bit 0 = block 15
4	Bit map blocks 16 - 23. Bit 7 = block 16, bit 0 = block 23
5	Bit map blocks 24 - 31. Bit 7 = block 24, bit 0 = block 31

2031 Directory Header - Track 18 Sector 00

Byte	Data	Definition
1-143		Reserved for 2031 BAM
144-161		Diskette name, padded with shifted spaces
162-163		Diskette ID Nr.
164	160	Shifted space
165-166	50, 65	ASCII '2a' identifies DOS version & format
167-170	160	Shifted spaces
171-255	00	Not used

4040 Directory Header - Track 18 Sector 00

Byte	Data	Definition
1-143		Reserved for 4040 BAM
144-161		Diskette name, padded with shifted spaces
162-163		Diskette ID Nr.
164	160	Shifted space
165-166	50, 65	ASCII '2a' identifies DOS version & format
167-170	160	Shifted spaces
171-255	00	Not used

Note: ASCII data may appear in bytes 180 - 191 on some diskettes.

8050 Directory Header - Track 39 Sector 00

Byte	Data	Definition
0-1	38-00	Track-Sector pointer to first BAM block
2	67	ASCII 'c' identifies DOS 2.5 format
3	00	Reserved for future DOS use
4-5		Not used
6-21		Diskette name, padded with shifted spaces
22-23	160	Shifted spaces
24-25		Diskette ID Nr.
26	160	Shifted space
27-28	50, 67	ASCII '2c' identifies DOS version & format
29-32	160	Shifted spaces
33-255	00	Not used

8250 Directory Header - Track 39 Sector 00

Byte	Data	Definition
0-1	38-00	Track-Sector pointer to first BAM block
2	67	ASCII 'c' identifies DOS 2.7 format
3	00	Reserved for future DOS use
4-5		Not used
6-21		Diskette name, padded with shifted spaces
22-23	160	Shifted spaces
24-25		Diskette ID Nr.

26	160	Shifted space
27-28	50, 67	ASCII '2c' identifies DOS version & format
29-32	160	Shifted spaces
33-255	00	Not used

D9060/D9090 Directory Header - Track 0 Sector 0

Byte	Data	Definition
0-1		Track-Sector pointer to Bad Track & Sector List
2-3	00-255	Identifies DOS 3.0 format
4-5	76-00	Track-Sector of first Directory Block
6-7	00-00	Not used
8-9	01-00	Track-Sector of first BAM Block

DIRECTORY BLOCK FORMATS - ALL DOS VERSIONS

2031 Directory Blocks - Track 18 Sectors 01 thru 18

4040 Directory Blocks - Track 18 Sectors 01 thru 18

8050 Directory Blocks - Track 39 Sectors 01 thru 29

8250 Directory Blocks - Track 39 Sectors 01 thru 29

D9060/D9090 Directory Blocks - Starting on Cylinder 76, uses all tracks - Sectors 00 thru 31, then expands to additional blocks as needed, providing 'unlimited' directory size.

Byte	Data	Definition
0-1		Track-Sector pointer to next directory block
2		File type
3-4		Track-Sector pointer to first file block
5-20		File name, padded with shifted spaces
21-22		Track-Sector of 1st side sector if RELative file
23		Record length if RELative file
24-27		Reserved for future file info
28-29		Track-Sector pointer for replacement
30-31		Number of blocks used by the file
32-255		Seven more 32-byte file entries (same as 2-31 above, plus two additional unused bytes)

Notes to Directory Block formats - all DOS versions:

1. 32 bytes per file entry, except the first entry is 30 bytes
2. Total of eight (8) file entries per directory block
3. File Type are:

Scratched files	$00
Sequential data	$01
Program files	$02
User-defined	$03
Relative Record	$04

4. File Type codes are OR'ed with $80 when file is properly closed
5. Track value of 00 in byte zero indicates the last used block in the directory. Sector value then shows next byte to use.

DISK DATA FILE FORMATS - All DOS Versions

Program Files

Byte	Definition
0-1	Track-Sector pointer to next program block
2-255	Up to 254 bytes of BASIC program text. End-of-file is marked by three consecutive bytes of $00.

Sequential and Relative Record Data

Byte	Definition
0-1	Track-Sector pointer to next sequential data block
2-255	Up to 254 bytes of data with carriage returns as terminators between data items

Notes: Track link of $00 in byte zero indicates last data block. Sector link is then next byte position to receive data. End of Relative Record data indicated by reading $ff.

Relative File Side Sector Format

Byte	Definition
0-1	Track-Sector pointer to next side sector
2	Side sector number - if 4040 or 8050 relative file Constant $FE - if DOS 2.7 or DOS 3.0 relative file
3	Relative Record length
4-5	Track-Sector pointer - 1st side sector
6-7	Track-Sector pointer - 2nd side sector
8-9	Track-Sector pointer - 3rd side sector
10-11	Track-Sector pointer - 4th side sector
12-13	Track-Sector pointer - 5th side sector
14-15	Track-Sector pointer - 6th side sector
16-255	Track-Sector pointers to 120 data blocks Total of 720 blocks (max. 182.8 K bytes) per file.

DOS 2.7 and DOS 3.0 Super Side Sector contains track/sector pointers to 127 groups of 6 side sectors as above for maximum file size of 23.25 Mb.

CHAPTER 7

DOS ERROR MESSAGES

DISK COMMANDS - QUICK REFERENCE

REQUESTING ERROR MESSAGES

The execution of the following program displays the error on the computer screen and resets the device error indicator:

BASIC 3.0	BASIC 4.0
OPEN 1,8,15	
INPUT#1,A,B$,C,D	PRINT DS$
PRINT A,B$,C,D	
INPUT#1,A,B$,C,D,E (Used with 8250 DOS 2.7 only)	
PRINT A,B$,C,D,E	

Where: A=message number, B$=error message, C=track, D=sector, E=drive nr. Error messages requested from the 8250 include drive number as a fifth variable. The BASIC 4.0 'PRINT DS$' automatically prints drive number.

SUMMARY OF CBM DISK ERROR MESSAGES

0	OK, no error exists.
1	Files scratched reponse. Not an error condition.
2-19	Unused error messages: should be ignored.
20	Block header not found on disk.
21	Sync character not found.
22	Data block not present.
23	Checksum error in data.
24	Byte decoding error.
25	Write-verify error.
27	Checksum error in header
30	General syntax error.
31	Invalid command.
32	Long line.
33	Invalid filename.
34	No file given.
39	Command file not found.
50	Record not present.
51	Overflow in record.
52	File too large.
60	File open for write.
61	File not open.
62	File not found.
63	File exists.
64	File type mismatch.
65	No block.
66	Illegal track or sector.
67	Illegal system track or setor.
70	No channels available.
71	Directory error,
72	Disk full or directory full.
73	Power up message, or write attempt with DOS mismatch.
74	Drive not ready.
75	Format Speed Error
76	Controller Error

DESCRIPTION OF DOS ERROR MESSAGES

NOTE: Error message numbers less than 20 should be ignored with the exception of 01 which gives information about the number of files scratched with the SCRATCH command.

20: READ ERROR (block header not found)
The disk controller is unable to locate the header of the requested data block. Caused by an illegal sector number, or the header has been destroyed.

21: READ ERROR (drive not ready) Indicates a hardware failure.

22: READ ERROR (data block not present)
The disk controller has been requested to read or verify a data block that was not properly written. This error message occurs in conjunction with the BLOCK commands and indicates an illegal track and/or sector request.

23: READ ERROR (checksum error in data block)
This error message indicates that there is an error in one or more of the data bytes. The data has been read into the DOS memory, but the checksum over the data is in error. This message may also indicate grounding problems.

24: READ ERROR (bad sector flag)
A hardware error has been created due to an invalid bit pattern in the data byte. This message may also indicate grounding problems.

25: WRITE ERROR (write-verify error)
This message is generated if the controller detects a mismatch between the written data and the data in the DOS memory.

27: READ ERROR (checksum error in header)
The controller has detected an error in the header of the requested data block. The block has not been read into the DOS memory. This message may also indicate grounding problems.

30: SYNTAX ERROR (general syntax)
The DOS cannot interpret the command sent to the command channel. Typically caused by an illegal number of file names, or pattern matching illegally used.

31: SYNTAX ERROR (invalid command)
The DOS does not recognize the command. The command must start in the first position.

32: SYNTAX ERROR (long line)
The command sent is longer than 58 characters.

33: SYNTAX ERROR (invalid file name)
Pattern matching is illegally used in the DOPEN or DSAVE command.

34: SYNTAX ERROR (no file given)
The file name was left out of a command or the DOS does not

recognize it as such. Typically, a colon (:) has been left out of the command.

39: SYNTAX ERROR (invalid command)
This error may result if the command sent to command channel (secondary address 15) is unrecognizable by the DOS.

50: RECORD NOT PRESENT
Result of disk reading past the last record via INPUT#, or GET# commands. This message will also occur after positioning to a record beyond end of file in a relative file. If the intent is to expand the file by adding the new record (with a PRINT# command), the error message may be ignored. INPUT or GET should not be attempted after this error is detected without first repositioning to a valid record number.

51: OVERFLOW IN RECORD
Data written with a PRINT# statement exceeds the defined relative record size. Data is truncated to the defined size. Typical cause is failing to include carriage returns sent as field or record terminators in calculating the record size.

52: FILE TOO LARGE
Record position within a relative file indicates that not enough blocks remain available on the disk to contain the specified number of records.

60: WRITE FILE OPEN
This message is generated when a write file that has not been closed is being opened for reading.

61: FILE NOT OPEN
This message is generated when a file is being accessed that has not been opened in the DOS. Sometimes, in this case, a message is not generated; the request is simply ignored.

62: FILE NOT FOUND
The requested file does not exist on the indicated drive.

63: FILE EXISTS
The file name of the file being created already exists on the disk.

64: FILE TYPE MISMATCH
The file type on a DOPEN command does not match the file type in the directory entry for the requested file.

65: NO BLOCK
This message occurs in conjunction with the B-A command. It indicates that the block to be allocated has been previously allocated. The parameters indicate the next higher track and sector number available. If the parameters are zeros then all higher numbered blocks are in use.

66: ILLEGAL TRACK AND SECTOR
The DOS has attempted to access a track or sector which does not

exist in the format being used. This may indicate a problem reading the pointer to the next block.

67: ILLEGAL SYSTEM T OR S
Special error message indicating an illegal system track or sector.

70: NO CHANNEL (available)
The requested channnel is not available, or all channels are in use. A maximum of five sequential files or three relative files may be opened at one time to the DOS. Direct access channels may have six opened files.

71: DIRECTORY ERROR
The BAM does not match the internal count. There is a problem in the BAM allocation or the BAM has been overwritten in DOS memory. To correct this problem, reinitialize the disk to restore the BAM in memory. Active files may be terminated by the corrective action.

72: DISK FULL
Either all blocks on the disk are used or the directory is at its limit. DISK FULL is sent when two blocks remain available to allow the current file to be closed.

74: DRIVE NOT READY
An attempt has been made to access an invalid device number or the disk is not powered-up or not up to speed.

75: FORMAT SPEED ERROR
While formatting diskettes the 8250 verifies that drive speed is within 2 milliseconds (1%) of being 200 milliseconds per revolution. If speed is outside that limit the formatting is halted with the disk error light on.

76: Controller error - a variety of conditions indicating controller hardware problems.

QUICK REFERENCE - DISK COMMANDS

The quick reference guide will assist the user in becoming familiar with the various commands as used in both BASIC 3.0 and BASIC 4.0, as well as with all Commodore disk units. Commands in BASIC 3.0 are upward compatible with BASIC 4.0. That is, if the user is familiar with BASIC 3.0, those commands will still work on computers furnished with BASIC 4.0.

DISK COMMANDS QUICK REFERENCE

BASIC 3.0	UNIVERSAL WEDGE	BASIC 4.0
SAVE "dr:fn",8	SAVE"dr:fn",8	DSAVE"fn",Ddr (drive defaults to 0)
LOAD"dr:fn",8	/dr:fn	DLOAD"fn",Ddr (drive defaults to 0)
LOAD"dr:*",8 RUN	^dr:fn (Runs program)	DLOAD"*" (Shifted Run/Stop key)
LOAD"$0",8 LIST (destroys memory)	>$0 (preserves memory)	DIRECTORY or DI<shifted R> (preserves memory)
10 OPEN1,8,15 20 INPUT#1,A,B$,C,D 30 PRINT A,B$,C,D	>return	?DS$ or ?DS (DS is number of error) (DS$ is error message)

NOTE: Assume that OPEN 1,8,15 has already been typed for all of the PRINT# commands in the following formats. Commands may be spelled out or abbreviated by the first letter as illustrated.

	FORMAT A DISK	
PRINT#1,"Ndr:dname,xx"	>Ndr:dname,xx	HEADER"dname",Ddr,Ixx
	INITIALIZE	
PRINT#1,"Idr"	>Ix	PRINT#1,"Idr"
	VALIDATE	
PRINT#1,"Vdr"	>Vdr	COLLECT Ddr
	COPY (all disk)	
PRINT#1,"Cddr=sdr"	>Cddr=sdr	COPY Dsdr TO Dddr
	COPY (single file)	
PRINT#1,"Cdr:dfn= dr:sfn"	>Cdr:dfn=dr:sfn	COPY Ddr,"sfn"TO Ddr,"dfn"
	CONCATENATE FILES	
PRINT#1,"Cdr:dfn= dr:sfn1,dr:sfn2,...	>Cdr:dfn=dr:sf1, dr:sfn2,...	CONCAT Ddr,"sfn"TO Ddr,"dfn"
	RENAME FILES	
PRINT#1,"Rdr:dfn=sfn"	>Rdr:dfn=sfn	RENAME Ddr,"sfn" TO "dfn"
	SCRATCH	
PRINT#1,"Sdr:fn"	>dr:fn	SCRATCH"fn",Ddr

PERMANENT ALTERATION OF DEVICE NUMBER

As assembled at the factory all CBM disk units have a device number of 8. This may be changed temporarily via the M-W command and will revert to 8 on power-up or reset. The device number may be changed permanently by means of modifications to printed circuit boards within the disk units. The hardware changes necessary differ for each model of disk unit.

W A R N I N G

These hardware modifications should be performed only by qualified CBM service technicians. Alterations attempted by unauthorized personnel will void the warranty on your disk unit.

CHANGE 2031

Two diodes (CR18 and CR19) control device number on the 2031. The diodes are located adjacent to I.C. chip U3J on the digital PCB. To change the device number, cut either one of the leads on one or both diodes as shown:

DEVICE NR.	CR18	CR19	
8	0	0	(0 = Unchanged)
9	0	1	(1 = Lead Cut)
10	1	0	
11	1	1	

CHANGE 4040 / 8050 / 8250

Three pins (22, 23, 24) on I.C. chip UE1 (on the digital PCB) control device number on these units. These pins are normally strapped to ground by the circuit etch. Three small circular blocks appear just to the left of UE1 - pin 22 is connected to the topmost of these blocks (when viewing the digital PCB from the front of the disk unit. To change device number either cut the appropriate trace(s) or remove UE1 and bend the correct pin(s) up so that they will not make connection when the chip is replaced.

DEVICE NR.	Pin 22	Pin 23	Pin 24	
8	0	0	0	(0 = Unchanged)
9	0	0	1	(1 = Cut/Bent)
10	0	1	0	
11	0	1	1	
12	1	0	0	
13	1	0	1	
14	1	1	0	
15	1	1	1	

CHANGE D9060 / D9090

Three pins (22, 23, 24) on I.C. chip 7G (on the topmost PCB) control the device number of these units. These pins are normally strapped to ground by the circuit etch. To change device number either cut the appropriate trace(s) or remove 7G and bend the correct pin(s) up so that they will not make connection when the chip is replaced.

DEVICE NR.	Pin 22	Pin 23	Pin 24	
8	0	0	0	(0 = Unchanged)
9	0	0	1	(1 = Cut/Bent)
10	0	1	0	
11	0	1	1	
12	1	0	0	
13	1	0	1	
14	1	1	0	
15	1	1	1	