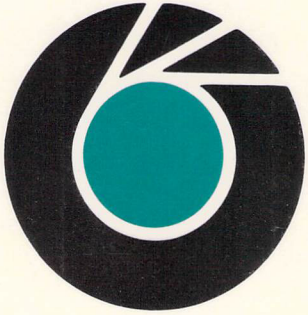
USER'S GUIDE

COMMODORE DISK DRIVE
MODEL BCD/128



BLUE CHIP

Disk Drives

RADIO INTERFERENCE

This equipmnet generates and uses radio frequency energy and if not installed and used properly, that is, in strict accordance with the manufacturer's instructions, may cause interference to radio and television reception. It has been type tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference in a residential installation. If this equipment does cause interference to radio or television reception, which can be determined by turning equipment off and on, the user is encouraged to try to correct the interference by one or more of the following measures:

- Reorient the receiving antenna
- Relocate the computer with respect to the receiver
- Mover the computer away from the receiver
- Plug the computer into a different outlet so that computer and receiver are on different branch circuits.

If necessary, the user should consult the dealer or an experienced radio/television technician for additional suggestions. The user may find the following booklet prepared by the Federal Communications Commission helpful: "How to Identify and Resolve Radio-TV Interference Problems." This booklet is available from the US Government Printing Office, Washington, DC20402, Stock No. 004-000-00345-4.

USER'S GUIDE

COMMODORE DISK DRIVE
MODEL BCD/128

Published by

BLUE CHIP ELECTRONICS, INC.

First edition 1986

Printed in Hong Kong

Every effort has been made to supply complete and accurate information in this manual. BLUE CHIP ELECTRONICS, INC. reserves the right to change Technical Specifications and Characteristics at any time without notice.

TABLE OF CONTENTS

PURPOSE

This manual has three objectives, namely to teach the reader the installation as well as the fundamental operation of the Blue Chip floppy disk drive and to enable him or her to manipulate programs or data on diskette used with the Commodore 64, 128, SX64, Plus 4, C16, or VIC-20 computer.

HOW TO USE THIS MANUAL

This manual is divided into eight chapters plus four appendices. Chapter 1 provides general information about initial set-up of the Blue Chip floppy disk drive together with the power and interface connections. Chapter 2 to chapter 7 describe the syntax and semantics of every command and statement in the disk operating system (DOS), intrinsic functions, and files handling. Chapter 8 tells you how to change the device number when more than one Blue Chip floppy disk drive is used with the computer. The appendices contain a list of error messages and codes, disk formats, the serial bus, and the specifications of the drive.

HOW TO DEAL WITH A NEW DISK

All new diskettes must be formatted before use. Type as following:

```
CLOSE 1 < RETURN >
OPEN 1,8,15,"N:diskname,id" < RETURN >
CLOSE 1 < RETURN >
```

Diskname is a user defined name not more than 16 characters whereas ID code is any 2 characters identifier.

CAUTION

The "new" command will erase all program that have been previously stored on the diskette. Applications programs, games, etc. that you purchase commercially should never be "newed" or formated. Each new diskette should be "newed" only once.

CHAPTER 1 INSTALLATION GUIDE

1.1 PACKING LIST

Your Blue Chip disk drive is securely packed in polyfoam. Save this packing for use when transporting your disk drive.

Check the contents of the box containing your disk drive. Besides this manual, you should find the following items:

1. Blue Chip disk drive
2. A serial cable
3. An AC adapter

1.2 INSTALLATION PROCEDURE

STEP 1 Make sure the power to the computer is turned off.

STEP 2 Take the head-protection card out of the slot and plug the serial bus cable into either one of the serial bus sockets on the back of the drive.

STEP 3 Plug the other end of the cable in the back of the computer.

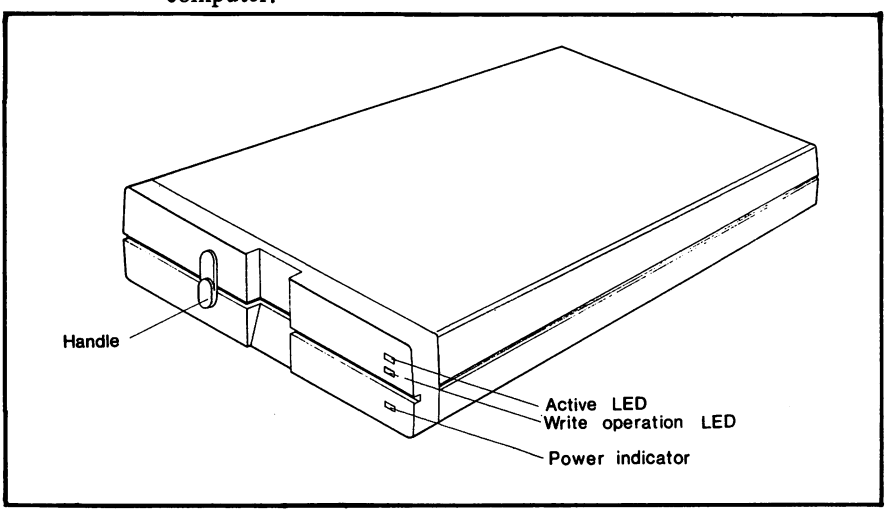


Figure 1 Front Panel

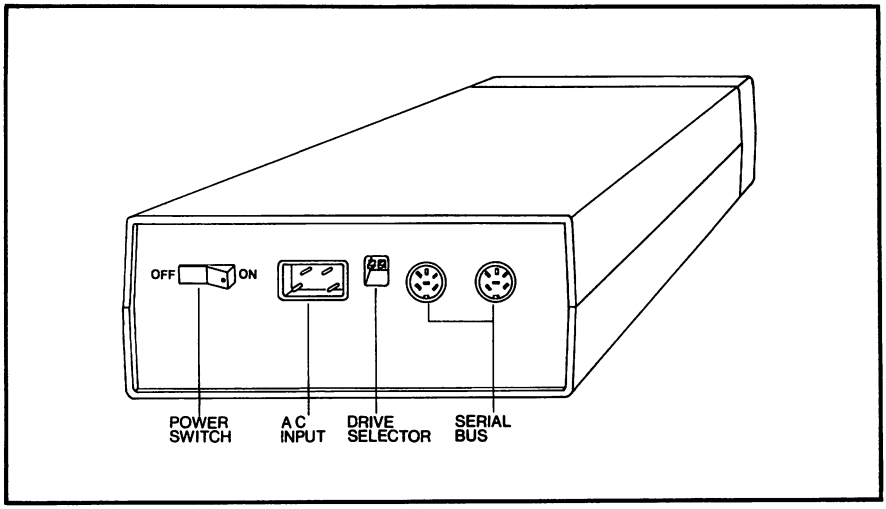


Figure 2 Back Panel

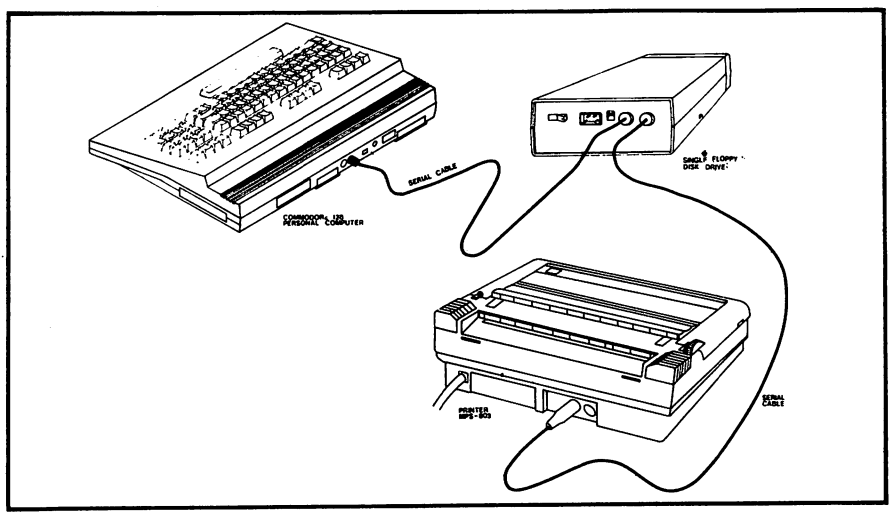


Figure 3 Floppy Disk Hookup

STEP 4 Plug one end of the AC adapter cable into the power socket at the back of the disk drive.

STEP 5 Plug the other end of the adapter cable into the wall socket.

1.3 DISKETTE HANDLING RECOMMENDATIONS

Since the recorded diskette contains vital information, reasonable care should be exercised in its handling. Longer diskette life and trouble-free operation will result if the following recommendations are followed.

1. Do not use a writing device which deposits flakes, e.g., lead or grease pencils, when writing on a diskette jacket label.

2. Do not fasten paper clips to diskette jacket edges.

3. Do not clean diskettes in any manner.

4. Do not touch a diskette surface exposed by jacket slot.

5. Keep diskettes away from magnetic fields and from ferromagnetic materials that may be magnetized.

6. Return diskettes to their envelope when they're not in use.

7. Protect the diskette from liquids, dust, and metallic substance.

8. Do not exceed the following storage environmental conditions:

TEMPERATURE	50°F to 125°F (10°C to 51.7°C)
RELATIVE HUMIDITY	8% to 80%

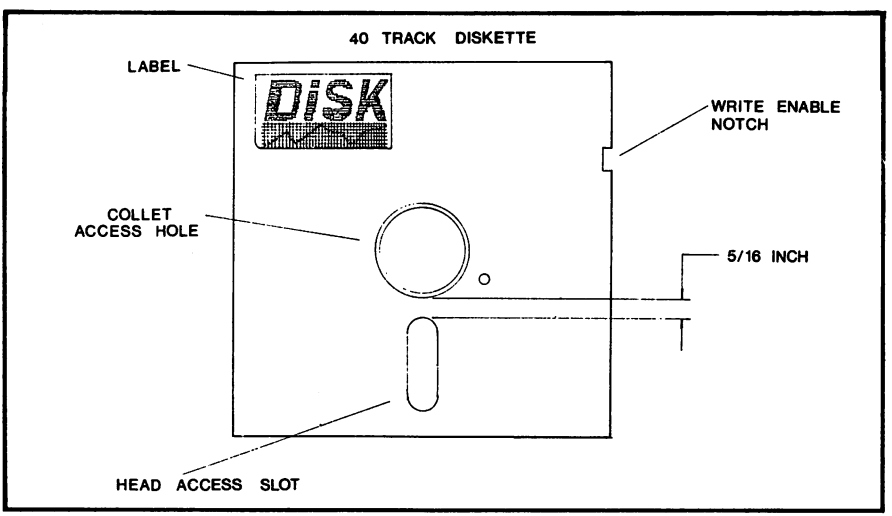


Figure 4 Identification fo 40 Track Diskette

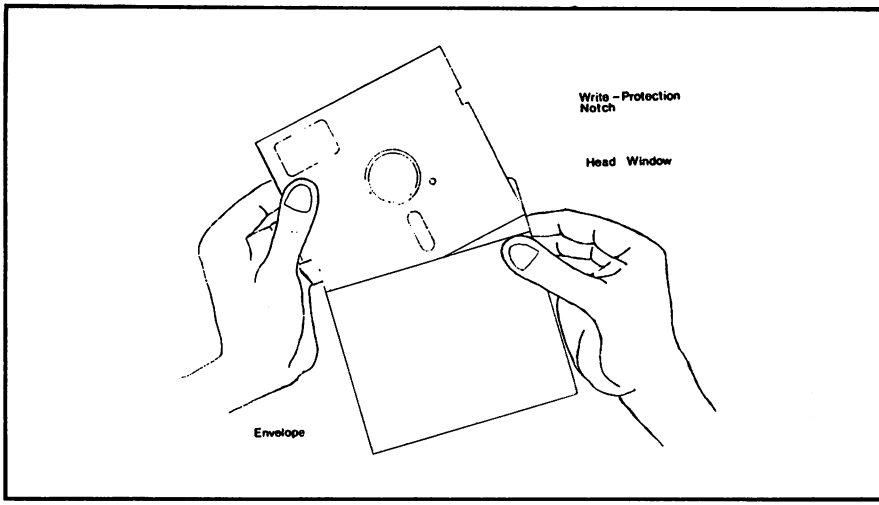


Figure 5 Diskette Handling

9. Diskettes should be stored in a box or cabinet when not in use.

10.Remove diskettes before turning the disk drive on or off.

1.4 FLOPPY DISKETTE LOADING

STEP 1 Move the door lever to the horizontal position.

STEP 2 Take the head-protection card out of the drive shot. Keep this card and inset it again before transporting your drive.

STEP 3 Be sure that the computer and all peripherals are properly connected, then switch on the power of the disk drive.

STEP 4 Remove the disketter from its storage envelope.

STEP 5 Be sure the write-protect slot in the jacket is covered, if the diskette is to be write-protected.

STEP 6 Carefully slide the diskette into the disk drive until the jacket is completely inside the disk loading slot.

STEP 7 Carefully turn the handle one quarter turn counter-clockwise to the vertical position across the diskette loading slot.

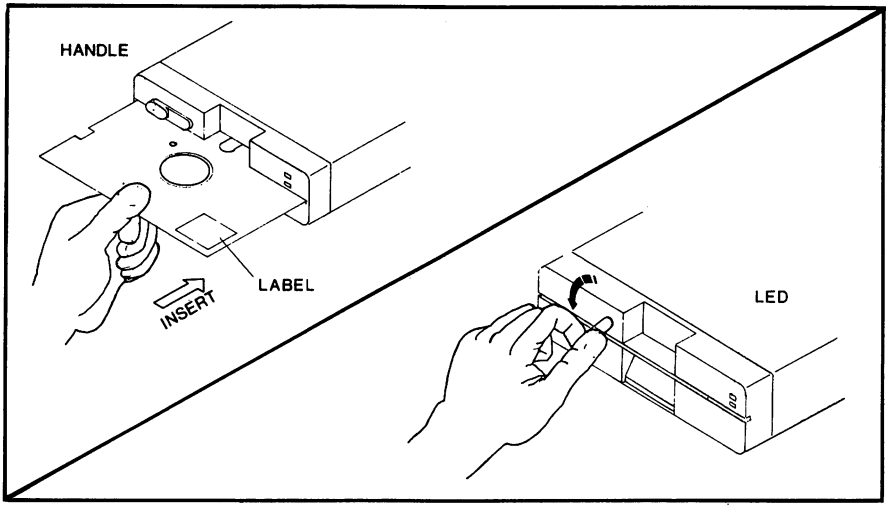


Figure 6 Inserting a Diskette

1.5 FLOPPY DISKETTE REMVOAL

STEP 1	Make sure the Active LED indicator is off before removing a diskette.
STEP 2	Turn the lever to the horizontal positon.
STEP 3	Carefully remove the diskette from the disk drive.

CHAPTER 2 OFTEN USED COMMANDS

The most frequently used commands are those commands that will load and save files on a disk that has already been formatted. Prepackaged programs fall into this category.

2.1 DOS COMMANDS AND STATEMENTS FORMAT

The DOS commands and statements described in the following chapters use the format given below.

FORMAT	Shows the correct format for the instruction.
PURPOSE	Tells why the instruction is used.
DESCRIPTION	Describes in detail how the instruction is used.
EXAMPLE	Shows a sample program or program segments that demonstrate the use of the instruction.
NOTE	Describes special cases and provides additional information.

2.2 PROGRAM FILE FORMAT

BYTE	DEFINITION
0-1	Track and block of next block in program file.
2-255	254 bytes of program information stored in CBM memory format (with key words tokenized). The end of the file is marked by three zero bytes.

WHAT IS THE COMMODORE BASIC

Commodore BASIC is the most extensive implementation of the BASIC lanuage available for commodore 64 and 128, SX64, C16, plus 4 and VIC-20 personal computers. Along with the vast array of commands and statements available in BASIC, which includes 32 powerful commands and 8 instrinsic functions that give you easy access to the disk drive. These commands let you store, retrieve and modify your programs and data on disk.

The table below lists the various commodore computers and the versions of BASIC each contains:

Computer	BASIC Version
PET 2000	1.0
VIC-20	2.0
C-64	2.0
CBM 3000	3.0
C-16	3.5
PLUS 4	3.5
CBM 8000	4.0
C-128	7.0

The version number is a measure of the power of the BASIC. For example, BASIC 4.0 is more powerful than BASIC 2.0.

2.3 DATA CHANNELS

When the data is sent, the specific channels is used for specific tasks.

There are 16 channels on the BCD drive. Usually only three or four of these can be used at one time. The following table shows the use of channels:

CHANNEL #	FUNCTION
0	LOAD
1	SAVE
2-14	FILE
15	COMMAND

2.4 LOAD / DLOAD / BLOAD

FORMAT

BASIC 2.0 / 7.0	LOAD "program $", device # [, command #]
BASIC 7.0	DLOAD "program $" [, D drive #, U device #] BLOAD "program $" , D drive #, U device # B bank #, P start #

PURPOSE

To load a program from the diskette into the computer 's memory.

DESCRIPTION

The program name can be a string variable or a filename that has been enclosed in quotes. The device number is preset on the disk drive's circuit board to be 8. If you have more than one drive, read chapter 8. This manual assumes that you are using 8 as the device number for the disk drive.

The command number is optional and can be a variable or a fixed value. If not given, or zero, the program is loaded normally beginning at the start of the BASIC memory of the computer. Specifying a one for the number will load the program to the same location that it was saved from. Unless specially instructed, prepackaged software does not need a command number.

In these commands, drive no. (ϕ or 1), device no. (8,9, 10, or 11), RAM bank no. (0-15) and start address location, which are preceded by letters D, U, B, and P, respectively.

EXAMPLE

```
LOAD "TEST", 8
DLOAD "TEST"
BLOAD "SPRITE", Dϕ, U8, Bϕ, P3584
```

NOTE

In addition to loading your program into the computer's current memory, LOAD erases any previous program located there.

2.5 SAVE / DSAVE / BSAVE

FORMAT

BASIC 2.0 / 7.0	SAVE "program $", device #
BASIC 7.0	DSAVE "program $" [, D drive #, U device #] BSAVE "program $" [,D drive #, U device # B bank #, P start # TO Pend #]

PURPOSE — To save a copy of the program in the computer memory onto the disk for later use.

DESCRIPTION — The command number is optional as in the LOAD command. The program will be saved from the start of the Basic memory area to the end of the program unless a file is already on the disk that has the same file name, or if there is not enough room on the disk. If either of the last two conditions occur, an error message will be displayed.

In these commands, drive no. (ϕ or 1), device (8,9, 10 or 11), RAM bank no. (0-15), start address location and end address location, which are preceded by letters D, U, B, P and TO P, respectively.

EXAMPLE

```
SAVE "FIRSTFILE", 8
DSAVE "FIRSTFILE"
BSAVE "SPRITE", Dϕ, U8, Bϕ, P3584 TO P4096
```

NOTE — To replace an already exsiting file with a revised version, use the following format:

SAVE "@0: program name", 8

This will replace any old program that has the same name with the program that is currently in the computer's memory.

2.6 VERIFY

FORMAT — VERIFY "program name", device#, command #

PURPOSE — To check the program currently in memory against the program stored on disk.

DESCRIPTION — This command does a byte by byte comparison of the programs, including line links which may be different for different memory configurations.

EXAMPLE — VERIFY "TEST 2", 8

2.7 DIRECTORY / CATALOG

FORMAT

BASIC 7.0	DIRECTORY [,D drive#] [,U device#] [Wildcard $] CATALOG [,D drive#] [,U device#] [Wildcard $]

PURPOSE — Displays the disk directory

Displays the disk directory contents but will not destroy the program in computer memory. Using wildcard, e.g. ?, * to display the directory you want.

EXAMPLE
DIRECTORY, Dϕ. U9
CATALOG "HELP*"

NOTE
If you use
LOAD "$", 8 : LIST
You will get the same display but the program in your computer will be erased.

2.8 RUN

FORMAT RUN [filename$]

PURPOSE To execute the program currently in memory.

EXAMPLE
```
RUN "ALEX"
   or
LOAD "ALEX", 8
SEARCHING FOR "ALEX"
LOADING
READY

RUN
```

2.9 BOOT

FORMAT

BASIC 7.0	BOOT ["programs $"] [,D drive#] [,U device#]

PURPOSE Loads and executes a program

DESCRIPTION This command loads and executes binary file at its starting address.

EXAMPLE BOOT (Boot and CP / M disk)

3.0 PATTERN MATCHING AND WILD CARDS

To simplify the loading procedure, pattern matching allows you to specify certain letters in the program name so the first program on the disk that matches your pattern is the one loaded.

Examples:
```
LOAD "*", 8        (LOADs last file on disk. If nothing has been
                    loaded, it loads the first file on the disk.)
LOAD "HI*", 8      (LOADs first file that starts wiht HI)
LOAD "HI??",8      (LOADs first file that has 4 letters and begins
                    with HI)
LOAD "I?EQ", 8     (LOADs first file that has 4 letters but could
                    be IHEQ, IPEQ, etc.)
```

The asterisk (*) tells the computer not to worry about the rest of the name while the question mark (?) acts as a wild card.

The above notations can also be used when loading the directory into current memory. This allows checking for a list of specific programs. The procedure is the same as above except for the addition of a "$:".

Example: LOAD "$:I?EQ", 8 (LOADs all file names in the directory that have the correct first, third and fourth letters)

CHAPTER 3 UTILITY COMMANDS

The Blue Chip disk drive contains a number of commands that are used to perform special tasks within the disk drive. These include formatting a new diskette so that programs can be stored on it, erasing a program from the diskette, or copying a program from one file to another. These operations are called utility commands. In order to use the utility commands, it is necessary to first establish a channel of communiction between the disk drive and the computer. This channel of communications allows the computer and the disk drive to exchange information.

3.1 CLOSE / DCLOSE

FORMAT

BASIC 2.0/7.0	CLOSE file#
BASIC 7.0	DCLOSE# file# [,U device#]

PURPOSE — Proper allocation of data blocks, close entry.

DESCRIPTION — The file number is that which was used in the OPEN statement. Once a file that has been opened, and is no longer needed for data transfer, it must be properly closed.

EXAMPLE

```
CLOSE 1
DCLOSE#1
```

NOTE — If you encounter an error message in a BASIC program, your files will be closed in BASIC, but not on the disk drive.

Immediately initialize the disk by typing the following command.

```
CLOSE 1: OPEN 1,8,15. "I":CLOSE 1.
```

3.2 COPY / BACKUP

FORMAT

BASIC 2.0	"Cϕ:copyname$ = ϕ:sourcename 1$ [,ϕ:sourcename 2$] .....
BASIC 7.0	COPY ["sourcename$"] [,D drive#] TO ["copynameϕ"] [, D drive#] [,U device#] BACKUP D sourcedrive# TO D copydirve# [,U device#]

PURPOSE — To copy any program or file to another program or film on the diskette.

DESCRIPTION — Copy is used to create a duplicate of an existing program or file on the diskette under a different name. Files can be copied onto the same diskette.

EXAMPLE — COPY "CHECKER" TO "BACKUP".

NOTE — It is possible to create a file that is the combination of several files using the COPY command as given below:

```
OPEN 1,8,15
PRNIT#1, "C 0:NEW=0:OLD1,0:OLD2,0:OLD3"
```

BACK UP command be used only with dual drive system.
The above command would combine OLD 1, OLD2 and OLD3 into one file on the diskette with the name NEW.

3.3 INITIALIZE

FORMAT "I [drive#]"

PURPOSE To revert the disk drive back to its original condition when it was powered up.

DESCRIPTION The INITIALIZE command should be used when the disk drive has encountered an error that prevents it from performing any further operations. This command restarts the computer. Any operations or files that were opened or in progress are terminated.

EXAMPLE OPEN 1,8,15, "I" or
OPEN 1,8,15:PRINT#1,"I"

NOTE An error condition on the disk will sometimes prevent you from performing an operation. INITIALIZE returns the disk drive to its original state when power is on.

3.4 NEW / HEADER

FORMAT

BASIC 2.0/7.0	"N[drive#]:diskname$ [,id$]"
BASIC 7.0	HEADER "diskname$" [,id$] [,D drive#] [,U device#]

PURPOSE (i) Formats new disk or re-formats used disk.

(ii) Clear the directory of an already formatted disk.

DESCRIPTION This command formats a disk to receive files. The disk name is user defined. The ID code is a 2 digit aplhanumeric identifier placed on the directory and on every block of the diskette. If you replace diskettes while writing data, the drive will know by checking the ID code.

EXAMPLE (i) Format disk:
Open 1,8,15:PRINT#1, "Nϕ:TEST DISK,A1"
HEADER"TEST DISK,A1"

(ii) Clear directory:
OPEN 1,8,15:PRINT#1, "Nϕ : TEST DISK"

NOTE — The INITIALIZE command should be used after you replace diskettes with different ID code.

3.5 OPEN

FORMAT — OPEN file #, device #, channel #,text$

PURPOSE — Creates a file by opening a communication channel between the computer and the disk drive.

DESCRIPTION — Any number from 1 to 255 can be used as file number. But numbers greater than 127 will cause the PRINT# statement to generate a linefeed after the return character. The device number is usually 8.

The channel number can be any number between 0 and 15, but 0 and 1 are reserved for the operating system for loading and saving. Channels 2 through 14 can be used to send data to files, and 15 is the command channel.

The text string is a character string that is printed to the file, as if with a PRINT# statement. If you attempt to open a file already opened, the error signal "FILE OPEN ERROR" will be generated.

EXAMPLE

```
OPEN 5,8,15,"TEST 1'     (creates a file TEST 1)
OPEN 1,8,15, "I"         (sends command to disk
                          on command channel)
```

3.6 PRINT#

FORMAT — PRINT#file#,text$

PURPOSE — Fills a previously opened file with data.

DESCRIPTION — The PRINT# command sends information to the disk drive.

EXAMPLE

PRINT#3,HS(fills file 3 with text string HS)

PRINT#1, "I" (sends disk command on command channel)

NOTE

It is possible to open the command channel without sending the text string message. The text string can be sent to the disk with the use of a PRINT# command. The following syntax is used:

PRINT#file#,text$

The file number must be opened before using this command or an error will result.

3.7 SCRATCH

FORMAT

BASIC 2.0	"Sϕ:programme 1$, ϕ:programme 2$....."
BASIC 7.0	SCRATCH "programme $"

PURPOSE

To erase a file or files from the diskette in order to get additional space on the diskette.

DESCRIPTION

More than one unwanted files can be deleted by using a single command, thus making room for new or larger files.

EXAMPLE

OPEN 1,8,15
PRINT#1, "Sϕ: HELLO" (erase file called HELLO)
PRINT#1, "Sϕ: HELLO, ϕ: GONE, ϕ: LET GO"
(erases files HELLO, GONE and LETGO)
SCRATCH "HELLO"

NOTE

Remember that only 40 characters at a time can be sent over the transmission channel to the disk drive.

3.8 RENAME

FORMAT

BASIC 2.0	"Rϕ:newname$ = oldname$"
BASIC 7.0	RENAME "oldname$" TO "newname$"

PURPOSE — To change the name of an existing file or program.

DESCRIPTION — RENAME lets you change the name· fo a file on the disk directory.

EXAMPLE —
```
OPEN 1,8,15
PRINT#1, "Rϕ: RENEWED = ϕ: ORIGINAL"
```

NOTE — RENAME will not work on any files that are currently opened.

3.9 VALIDATE / COLLECT

FORMAT

BASIC 2.0	"Vϕ"
BASIC 7.0	COLLECT [,D drive#] [,U device#]

PURPOSE — Removes wasted space on the disk.

DESCRIPTION — The VALIDATE command organizes the disk for maximum data storage.

EXAMPLE — OPEN 1,8,15, "Vϕ"

NOTE — VALIDATE erases random files. If your disk contains random files, DO NOT use this command!

3.10 DCLEAR

FORMAT

BASIC 7.0	DCLEAR [,D drive#] [,U device#]

PURPOSE — Closes all open channel on drive.

DESCRIPTION — This command closes and clears all open channels on the specified device.

EXAMPLE — DCLEAR

NOTE — In C-64 system
FOR I=1 TO 15 : CLOSE I : NEXT I

3.11 READING THE ERROR CHANNEL

To read the error channel, type:

```
PRINT DS
   or
10  OPEN 1,8,15
20  INPUT#1, A, B$, C, D
30  PRINT A,B$,C,D
40  CLOSE 1
50  END
```

After performing an INPUT# from the command channel, you get 4 variables that describe the error condition. The first, third, and fourth are numbers so numeric variables can be used. The inputs are organized as follows:

FIRST:	error number (0 means no error)
SECOND:	error description
THIRD:	track number where error occured.
FOURTH:	block (sector) in track where error occured.

CHAPTER 4 SEQUENTIAL FILES

Sequential files are those files that are stored and read, from beginning to end. There are three different types of sequential files that can be used. The first type is the program file which is abbreviated in the directory of the diskette as PRG. The program file is the only type of sequential file that can be loaded. The second and third types are designated as sequential (SEQ) and user (USR) respectively. These two types of files are used for data handling.

4.1 SEQUENTIAL FORMAT

BYTE	DEFININTION
0-1	Track and block of next sequential block
2-255	254 bytes of data with carriage return as record terminators

4.2 GET

FORMAT GET#file#, variable list

PURPOSE To retrieve data from the disk byte by byte.

DESCRIPTION GET# is used to load data into the computer memory byte by byte, including all seperators. This command should be used with character string variables in order to avoid error messages.

EXAMPLE
```
GET#2, A$
GET#A (only works for numerical data)
GET#A,X$,Y$,Z$ (gets variables X$, Y$ and Z$)
```

NOTE The GET# statement is very useful when the actual data content or structure is not known, such as a file on a disk that has been damaged. For situations when a group of data is to be read and the data structure is known, the INPUT# statement is better suited. But to examine the data in an unfamiliar or damaged file, the following program example can be used.

```
10 OPEN 3,8,6,"TEST"
20 GET#3, A$:PRINT A$;
30 IF ST=0 THEN 20 (ST is a status signal)
40 CLOSE 3
50 END
```

4.3 INPUT#

FORMAT — INPUT#file#, variable list

PURPOSE — Retrieve disk data in groups.

DESCRIPTION — The file number is the file number that was used in the OPEN command. The variable can reprint character strings or numbers. In order to read a group of data, it is necessary to be able to recognize the beginning and the end of that group of data. This is accomplished by using separators as explained in the PRINT# command. Numbers are stored with a space in front of them, which is empty for postive numbers and contains a negative sign for negative numbers.

EXAMPLE

```
10 OPEN 4,8,4,"@0:TESTFILE,S,W"
20 FOR N=1 TO 10
30 PRINT#4, N
40 NEXT N
50 CLOSE4
60 OPEN 3,8,3,"TESTFILE"
70 INPUT#3, P:PRINT P
80 IF ST=0 THEN 70
90 CLOSE 3
100 END
```

This example will write numbers 1 through 10 to a segment file called TESTFILE. Lines 70 and 80 read the data from the disk and print it out.

4.4 PRINT#

FORMAT — PRINT#file# data list (no space allowed between PRINT and #)

PURPOSE — For writing data to the diskette.

DESCRIPTION — The PRINT# command is used to direct any output to the file that has been opened. The file number is that of the file that has been opened by the OPEN command. The data consists of variables and/or text that has been enclosed inside of quotation marks. The PRINT# command works like the PRINT command in BASIC. Commas used to seperate items will cause spaces to be stored on the disk. Semicolons will keep spaces from being stored. If both commas and semicolons are absent, a carriage return (CR) will be stored at the end of the data that is written.

EXAMPLE

```
10 A$ = "THIS IS"
20 B$ = "A TESTING"
30 OPEN 2,8,7 "0:Test 1,S,W"
40 PRINT#2, A$, B$"PROGRAM"
50 CLOSE 2
60 END
```

The data and its position on the disk would look like this:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
T H I S   I S                                  A

20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35
T E S T I N G P R O G R A M CR eof
```

*eof – end of file

NOTE — When using the PRINT# statement, if you use commas to seperate items on the line, the items will be seperate items on the line, the items will be seperated by blank spaces. Semicolons, however, do not result in any blank space.

4.5 OPEN / DOPEN

FORMAT

BASIC 2.0	OPEN file#, device#, channel#, "[@] ϕ:filename $, type $, direction $"
BASIC 7.0	DOPEN#file#, "filename $" [,D drive#] [,U deviec#]

PURPOSE — Opens a sequential file

DESCRIPTION — The file number is the same as in previous uses of the OPEN command, the device number is usually 8, the channel number is a data channel, 2 through 14. The name is the file name, for which no wild cards or pattern matching may be used if you are creating a write file. The type can be any type given in the chart below, or at least the first letter of each type. The direction must be READ or WRITE, or at least one of their first letters. @ means OPEN–WITH–REPLACE.

FILE TYPE	MEANING
PRG	program file
SEQ	sequential file
USR	user file
REL	relative (not implemented in BASIC 2.0)

[@] Means OPEN–WITH–REPLACE	
Type	
P	Porgram file
S	Sequential file
U	User file
R	Relative file
Direction	
W	Write to disk
R	Read from disk
M	Modify in byte way

EXAMPLE

OPEN 3,8,3, "0:DATA,S,R"
OPEN A,B,C, "0:WORK,P.W"
OPEN A,B,VC, "0:"+A$+"U,R" (opens a read file with a name specified by the string variable A$)
OPEN 6,8,6 "@0:TELCO,S,W" (replaces old version of the file with a new one)

NOTE

Sequential files are stored and read sequentially from beginning to end. There are basically three different types of sequential files that can be used. The first is program file (PRG), the only sequential file that can store and read programs. The second type, sequential (SEQ), and the third type, user (USR), are for data handling. These two types of files must be opened just like the command channel in the last chapter.
Once a file has been opened for reading or writing, the three commands – PRINT#, INPUT#, and GET# can be used to actually transfer the data.

4.6 APPEND

FORMAT

BASIC 2.0	OPEN file#, device#, channel#, "ϕ:filename $, A"
BASIC 7.0	APPEND#file#. "filename $" [,D drive#] [;U deivce#]

PURPOSE

This command allows to reopen an existing file and add information to the end of it.

ST : Status Variable		
0	OK	
1	TIME OUT ON TALKER	
2	TIME OUT ON LISTENER	
4	DATA BLOCK TOO SHORT	(CASSETTE)
8	DATA BLOCK TOO LONG	(CASSETTE)
16	VERIEY ERROR	(CASSETTE)
32	CHECKSUM ERROR	(CASSETTE)
64	E0I (End-Of-File)	
128	Device not present	

4.7 CONCAT

FORMAT

BASIC 7.0	CONCAT "attach file $" TO "head files $" [,D drive#] [,U device#]
PURPOSE	Concatenates two data files.
DESCRIPTION	This command attaches the attach file $ to the end of headfile $ and retains the name of head file $.
NOTE	Keep the filenames length within ten Characters for the drive limited buffer capacity.
EXAMPLE	CONCATE "LAST" TO "ACQ".

CHAPTER 5 RANDOM ACCESS FILES

Sequential files are handy when working with continuous streams ot data and program files, but quite often it is desirable to be able to access certain data without having to go sequentially through each byte. For this type of data, it is better to use a random access method of data storage and retrieval. Random files provide the capability of obtaining data from within a file without reading the entire file. Relative files will be discussed in CHAPTER 6. The decision to use random files versus relative files is determined by considering several differences between the two methods. Random files are more suitable when speed is a major consideration. Many machine language programs untilize random files because of its speed advantage over relative files. The problem with random files is that the location of the data must be maintained by the program using the random file while relative file locations are maintained by the disk operating system (DOS). This is the primary cause for the difference in speed. Random files can be more easily removed from the diskette since the DOS does not maintain those files.

Random files are files that have been written to a certain designated location on the diskette. From the definitions given in APPENDIX B, the disk is physically divided into 35 tracks with each track containing from 17 to 21 sectors or blocks.

BLOCK DISTRIBUTION BY TRACK

Track number	Range of Sectors	Total # of Sectors	
1 to 17	0 to 20	21	single sided
18 to 24	0 to 18	19	
25 to 30	0 to 17	18	
31 to 35	0 to 16	17	
35 to 52	0 to 20	21	double sided
53 to 59	0 to 18	19	
60 to 65	0 to 17	18	
66 to 70	0 to 16	17	

5.1 BLOCK-ALLOCATE

FORMAT "B–A"; drive#; track# ; block#

PURPOSE To determine if a particular block is free on the disk all allocate it if it is free.

DESCRIPTION The user can make sure a particular block is available by using the BLOCK-ALLOCATE command. This allows the use of block commands on a disk with files already on it. By checking the BAM, the command determines if the specified block has been used. Files can be maintained since the BAM updates each time a file is stored on the disk. Blcok commands do not update the BAM and so will not be recognized unless a BLOCK-ALLOCATE has been executed.

If B-A determines that the specified block has already been used, an error signal (6.5) will be generated. The error message shows the numbers of the next available track and block on the disk. This block does not get allocated, so the B-A command must be used again, but this time you can be sure that the block specified if free to use:

EXAMPLE

```
10 OPEN 15,8,15:OPEN 6,8,6,"#"
20 PRINT#6, "THE BLOCK DATA"
30 T=3:S=2
40 PRINT#15, "B-W 6 ϕ"; T ; S
50 INPUT#15, A, B$, C, D
60 IF A=65 THEN T=C:S=D:GO TO 40
70 PRINT#15, "B-A ϕ" ; T ; S
80 PRINT "DATA WRITTEN IN TRACK:"T,
   "SECTOR:"S
90 CLOSE 6:CLOSE 15
100 END
```

This program will allocate a block and write to that block. If the block is already used, it will write to the next available one, as shown by the error message.

NOTE The VALIDATE command does not recognize random files and should never be used on a disk that has random files.

5.2 BLOCK-FREE

FORMAT "B–F"; drive#; track#; block#

PURPOSE To make a specified block on the diskette available for use.

DESCRIPTION B-F is the opposite of B-A. If a block is not used anymore, B-F frees the block by erasing its entry in the BAM.

EXAMPLE
```
10 OPEN 5,8,5,"#"
20 OPEN 15,8,15:PRINT#15, "B-F 0 7 6"
30 CLOSE 5:CLOSE 15
   (free track 7, block 6 for use)
```

5.3 BLOCK-READ

FORMAT "B-R"; channel#; drive#; track#; block#

PURPOSE To read a specific block of data from the disk.

DESCRIPTION The file number and the channel number are the ones that have been opened. The track number and the block number indicate which 256 byte block is to be read. Executing this command causes the disk drive to move the specified block of data into the buffer area. The data can be read from the buffer area using the INPUT# and GET# commands. Only data that has been stored on that particular block will be read. Any unused bytes in the block will not be read.

EXAMPLE
```
10 OPEN 15,8,15
20 OPEN 8,8,8,"#"
30 PRINT#15, "B-R 8 ϕ 5 9"
40 GET#8,A$
50 PRINT A$;
60 IF ST=0 THEN 40
70 PRINT "READ COMPLETE"
80 CLOSE 8:CLOSE 15
```

To read the content of block 9 on track 5 and display the block's content on the screen.

5.4 BLOCK-WRITE

FORMAT "B-W"; channel#; drive#; track#; block#

PURPOSE To write a block of data to a specific location on the disk.

DESCRIPTION The B-W command causes the previously stored buffer information to be written to the specified location on the disk. The information is transferred to the buffer through the channel other than the command channel that has been opened. This transfer is accomplished with the PRINT# command. The DOS deeps track of how many bytes are stored in the buffer and stores the byte count in the first byte of the block when B-W is executed. This means that only 255 bytes can actually be written to or read from the block. The pointer takes up the first byte of the block.

EXAMPLE

```
10 OPEN 15,8,15
20 OPEN 8,8,8,"#"
30 FOR B=1 TO 32
40 PRINT# 8, "ONTEST";
50 NEXT
60 PRINT#15,"B-W 8 ϕ 6 7"
70 CLOSE 8:CLOSE 15
```

A routine that will write data to block 6, track 7.

5.5 BUFFER-POINTER

FORMAT "B-P"; channel#; location#

PURPOSE To allow random access inside a block.

DESCRIPTION The B-P records the location of the last piece of data as well as where the next piece of data will be read. Random access to individual bytes inside a block is made possible by altering the pointer's location in the buffer.

EXAMPLE PRINT#15, "B-P 6 23" (sets pointer to the 23rd character in the buffer)

5.6 OPEN

FORMAT — OPEN file#, device#, channel#,"#[buffer#]"

PURPOSE — Opens a data channel for random access.

DESCRIPTION — A minimum of two channels must be opened in order to use random files. The command channel, channel 15, is opened to send commands and a data channel (2 to 14) is used for the data transfer. The data channel for random access files is opened by selecting the pound sign, "#", as the filename.

EXAMPLE

```
OPEN 5,8,15,"#"          (Don't care which buffer)
OPEN Y,Q,J, "#4"         (Specify Buffer #4)
```

5.7 USER 1

FORMAT — PRINT#file#,"U1";channel#;drive#; track#; block#

PURPOSE — To read a full 256-bytes block from disk to buffer.

DESCRIPTION — USER1 can be shortened to U1 or UA. This command works like the BLOCK-READ command except that USER1 forces the buffer-pointer to 255 in order to read the entire block of data.

EXAMPLE

```
10 OPEN 1,8,15
20 OPEN 2,8,2,"#"
30 PRINT#1,"U1 2 0 3 6"
40 GET#2,A$:PRINT A$;
50 IF ST=0 THEN 40
60 CLOSE 2:CLOSE 1
70 END
```

Program to get the entire 256 bytes from track 3 block 6 and display it on the screen

NOTE — USER commands, U1 and U2, are generally designed to work with machine language.

5.8 USER2

FORMAT PRINT#file#,"U2";channel#;drive#;track#;block#

PURPOSE To write a block of data on the disk without altering the buffer-pointer on the diskette.

DESCRIPTION USER2 can be shortened as U2 and UB. It is similar to the B-W command except it does not change the location of the buffer-pointer when the buffer is written on the disk. This is useful if you want to read a block of data into the buffer and modify it. After finding the particular data by the buffer-pointer and modifying it, you can use the USER2 command to rewrite the data on the disk and the buffer-pointer will be in the correct position. If B-W was used, the buffer-pointer would have to be reset first.

EXAMPLE

```
10 OPEN 1,8,15
20 OPEN 2,8,2,"#"
30 PRINT#1,"U1 2 0 18 3"
40 PRINT#1,"B-P 2 8"
50 PRINT#2,"A"
60 PRINT#1,"U2 2 0 18 3"
70 CLOSE 2:CLOSE 1
80 END
```

Line 30 reads track 18 block 3 into the buffer.
Line 40 moves the buffer-pointer to byte 8.
Line 50 changes byte 8 to the character"A".
Line 60 prints the buffer back to the disk.

Even though the buffer-pointer has been altered, USER2 makes sure the old buffer-pointer is not changed on the disk.

5.9 EXAMPLES OF RANDOM FILE

The problem with random files is that you have no way of keeping track of the blocks you have used. To keep track, the most common method is to create a sequential file to go with each random file. This file is used to keep just a list of record, track and block locations. This means you have 3 channels open to the disk for each random file. The command channel, the channel for the random data, and the channel for the sequential file. You are also using 2 buffers at the same time.

The following four programs use random access within blocks.

```
 90  REM--------SEQUENTIAL FILE DEMONTRATION----
 91  REM
 92  REM
 93  REM
100  REM CREATE SEQ FILE
110  OPEN3,8,3,"@0:SEQ DEMO,S,W" : REM  OPEN A BRAND NEW FILE
120  FOR I=1 TO 4:READ T$         : REM TYPE @ TO AVOID
                                    ERROR
130  PRINT#3,T$                   : REM STORE 4 DATA FIRST
140  NEXT I
150  CLOSE 3
200  PRINT"READ SEQ DEMO CONTENTS":GOSUB 1000
301  REM
302  REM
303  REM
305  PRINT"ADD REC TO SEQ FILE, < Q > UIT"
310  OPEN5,8,5,"0:SEQ DEMO,S,A"  : REM OPEN FILE FOR APPEND
320  INPUT T$:IF T$="Q" THEN CLOSE 5:GOSUB 1000:END
330  PRINT#5,T$
340  GOTO 320
900  REM
910  REM
920  REM
999  REM READ SEQ FILE
1000 OPEN 4,8,4,"0:SEQ DEMO,S,R"  : REM OPEN FILE FOR READ
1010 INPUT# 4,T$:PRINT T$         : REM READ DATA FROM FILE
1020 IF ST=0 THEN 1010            : REM TEST END OF FILE
1030 PRINT "EOF"
1040 CLOSE 4
1050 RETURN
1100 REM
1200 REM
1300 REM
2000 DATA HELLO, HOW, ARE, YOU : REM THE FIRST 4 DATA
```

CHAPTER 6 RELATIVE FILES

Relative files allow you to work on any piece of data on the disk like a random file and you do not have to maintain the files in your own programs. It is very convenient for data handling especially when you structure your files into records and into fields within those records.

The DOS keeps track of the tracks and sectors (blocks) used. It even allows records to overlap from one block to the next. It does this by establishing side sectors, a series of pointers for the beginning of each record. A file can have up to 6 side sectors and each side sector can point to a maximum of 120 records, i.e., a file may have a total of 720 records. As each record can be 254 characters long, one file can fill the entire disk.

The first tow bytes of the data block specify the track and sector of the next data block. The remaining 254 bytes contain the actual data. The side sectors are used for reference to all side sector locations and the 120 data block locations related to that side sector.

6.1 RELATIVE FILE FORMAT

DATA BLOCK:	
BYTE	MEANING
0.1	Track and sector of next data block.
2-255	Empty records – FF in the 1st byte – all 00 in the rest of the area Partially filled records – paddled with nulls or 00

SIDE SECTOR BLOCK	
BYTE	DEFINITION
0.1	Track and sector of next side sector block.
2	Side sector numbers (0-5)
3	Record length
4,5	Track and sector of 0th side sector.
6,7	Track and sector of 1st side sector.
8,9	Track and sector of 2nd side sector.
10,11	Track and sector of 3rd side sector.
12,13	Track and sector of 4th side sector.
14.15	Track and sector of 5th side sector.
16-255	Track and sector pointers up to 120 data blocks.

6.2 OPEN A RELATIVE FILE

FORMAT

BASIC 2.0	OPEN file#, device#, channel#, "filename $, L," + CHR$ (record length)
BASIC 7.0	DOPEN#file#, "filename $", L record length [,D drive L] [,U deviceL]

PURPOSE

(i) To create a relativ file when it is first opened.

(ii) To open existing relative file.

DESCRIPTION

Relative file are created when they are first opened. The same file will be used until it is closed. A relative file can only be erased from a disk by using the SCRATCH command or by reformatting the entire disk. The "@" sign, used with SAVE and REPLACE, will not work with relatvie files.

EXAMPLE

(i) (a) OPEN 2,8,2,"0:FILE,L"+CHRS(100)
(record length is 100)

(b) OPEN G,8,G, "0:"+A$+, +CGR$(P)

(ii) OPEN 4,8,4,"0:REL"

NOTE

In order to read or write, BEFORE ANY OPERATION, you must position the file pointer to the correct record position.

6.3 RECORD / POSITION

FORMAT

BASIC 2.0	"P" + CHR$ (96+REL channel) + CHR$ (Low-byte Record Number) + (Hi-byte Record Number) + CHR$ (Offset)
BASIC 7.0	RECORD#file#, record number# [, offset]

PURPOSE — To position the file pointer at a record.

DESCRIPTION — As the largest number one byte can hold is 256, two bytes must be used to specify the position of 720 available records. The rec#HI holds the significant part of the address and the rec#LO the least signigicant one. The relationship is given by rec#=rec# HI*256+rec#LO where rec# is the actual position in a record where data transfer starts.

EXAMPLE

```
OPEN 1,8,15
PRINT# 15,"P"CHR$(2)CHR$(1)CHR$(0)
PRINT# 15,"P"CHR$(CH)CHR$(R1)CHR$(R2)CHR$
(P)
```

6.4 EXAMPLE OF RELATIVE FILE

```
  1 REM---------- RELATIVE FILE DEMONTRATION-----
  2 REM
  3 REM
  8 REM CREATE A REL FILE
 10 OPEN 2,8,3, "REL DEMO,L," + CHR$(12) :
    REM CREATE REL FILE WITH RECORD LENGTH = 12
 21 REM
 30 FOR I=1 TO 20:READ T$        : REM READ THE FIRST 20 DATA
 40 PRINT#2, T$                  : REM BY REL'S OWN CHANNEL
 50 NEXT I
 60 CLOSE 2:CLOSE 5              : REM CLOSE FOR CREATE
                                   FINISH
 70 GOSUB 2000                   : REM READ BACK
100 END
190 REM
```

```
1900  REM
1910  REM
1920  REM
1930  REM
2000  PRINT"DISPLAY RELATIVE FILE CONTENTS"
2110  OPEN 2,8,2,"REL DEMO"          : REM OPEN FOR READ
2115  INPUT"FROM WHERE";A
2120  RECORD #2, A                   : REM SETUP RECORD POINTER
2130  FOR I=A TO 20
2140  INPUT #2, T$                   : REM READ BACK DATA
2145  PRINT T$;"  ";                 : REM BY REL'S OWN CHANNEL
2150  NEXT I
2160  CLOSE 2: CLOSE 5               : REM CLOSE FOR READ OK
2200  RETURN
3000  REM
3100  REM
3110  REM
5000  DATA THIS, IS, A, COMPUTING, MACHINE, AND, THAT
5010  DATA ELECTRONIC, DEVICE, IS, VERY, POWERFUL
5030  DATA IN, NUMBER, CRUNCHING, UNDER, SIGNAL, PROCESSING
5040  DATA DO, YOU, SATISFY, IT
```

CHPATER 7 COMMANDS FOR DOS ROUTINE

Routines can be designed that reside and operate on the disk controller, DOS routines can be added that come from the diskette. Routines can be added the same way as the DOS Support Program is "wedged" into you memory.

7.1 BLOCK-EXECUTE

FORMAT "B-E";channel#; drive#; drive#; track#; block#

PURPOSE Executes a particular block of machine language routine in disk memory.

DESCRIPTION BLOCK-EXECUTE loads a machine language routine and executes starting at location 0 in the buffer. The execution will continue until an RTS (return from surbroutine) command is encountered.

EXAMPLE PRINT#3,"B-E";6;0;17;3

7.2 MEMORY-EXECUTE

FORMAT PRINT#file#,"M-E" CHRS(low address byte) CHR$ (high byte)

PURPOSE Executes program in disk memory.

DESCRIPTION With this command, any routine in the DOS memory can be executed.

EXAMPLE PRINT#8,"M-E"CHR$(0)CHR$(4)

7.3 MEMORY-READ

FORMAT — PRINT#file#,"M-R"CHR$(low byte address)CHR$ high byte)

PURPOSE — Read data from drive memory.

DESCRIPTION — This command reads a byte of data from the disk dirve memory. The byte of data is in the location specified by the low and high bytes of the location address.

EXAMPLE — PROGRAM TO READ THE DISK CONTROLLER'S MEMORY

```
10  OPEN 15,8,15
20  INPUT "LOCATION=";A
30  FOR L=1 TO 50
40  AH=INT(A/256):AL=A-AH*256
50  PRINT#15,"M-R" CHR$(AL)CHR$(AH)
60  GET#15,A$
70  PRINT ASC(A$+CHR$(0 ))
80  A=A+1
90  NEXT
100 INPUT "CONTINUE";A$
110 IF LEFT$(A$,1)="Y" THEN 30
120 GOTO 20
```

NOTE — Using INPUT# to access the error channel might lead to peculiar results, but cleaning up can easily be done by using any command other than a memory command.

7.4 MEMORY-WRITE

FORMAT

PRINT#file#,"M-W" CHR$(address low byte) CHR$(address high byte)CHR$(# of characters) CHR$(data)

PURPOSE

To write data into the disk controller's memory.

DESCRIPTION

MEMORY-WRITE enables the storing of a maximum of 34 bytes of data into the disk drive controller's memory using one command. Both the M-E and USER commands can be used to execute this code.

EXAMPLE

```
10 OPEN 1,8,15
20 PRINT#1,"M-W" CHR$(0)CHR$(112)CHR$(3)
   CHR$(169)CHR$(8)CHR$(96)
30 CLOSE 1
```

This routine writes three bytes to locations 7000 H 7001H, and 7002H. (256*112+0=28672= 7000 H) The three bytes are 169 (A9H, a PAGE ZERO instruction), 8 (8H, a location), and 96 (60 H, a RETURN instruction). When executed, this program would cause the disk controller to load its accumulator with the contents of location 0008H and then return control back to the disk drive.

7.5 USER COMMAND

FORMAT "U" + Command $

PURPOSE Executes a particular location in the drive memory or act as brust instruction.

DESCRIPTION Format instruction
"Uφ"CHR$(first setting) CHR$ (second setting) CHR$ (interval/ID1) CHR (sector size code/ID2)
CHR$ (last track) CHR$ (sector byte code) CHR$ (first track)
CHR$ (physical first track) CHR$ (Fill byte) CHR$ (Sector Tables)

BIT		7	6	5	4	3	2	1	φ
First Setting	0	Normal format	Soft sector	Single side	Side φ	φ	φ	1	φ
	1	Partial format	Index hard sector	Double side	Side 1	φ	φ	1	φ
Second Setting	0	Commodore format	Create nornal table	Smallest sector number on track					
	1	IBM format	Specified sector table						
Interval / ID1		Sector interval – 1 (for IBM format) or first ID code (for commodore format)							
Interval / ID2		00-128 byte/sec 01-256 byte/sec (for IBM format) or Second ID code 02-512 byte/sec (Default 256 byte/sec) (for Commodore format) 03-1024 byte/sec							
Last track		Last logical track (default 39)							
Sector byte code		Largest sector number on track code (default 256 byte/sec)							26-128 byte/sec 16-256 byte/sec 9-512 byte/sec 5-1024 byte/sec
First track		First logical track (default φ)							
Physical first track		First physical track (default φ)							
Fill byte		Fill this byte for empty (default $ E5)							
Sector table		Active only if bit 6 of second setting (specified sector table) is set							

The format instruction must be followed by INQUIRE DISK or QUERY DISK FORMAT

Read and Write Instruction

"Uϕ" CHR$ (Control) CHR$(Track) CHR$ (Sector) CHR$ (Sector/Track) [CHR$ (Next Track)]

BIT		7	6	5	4	3	2	1	ϕ
Control	ϕ	Transfer buffer to computer	Report error	Read/Write be active	Side ϕ	ϕ	ϕ	READ	ϕ
	1	Don't transfer buffer to computer	ignore error	buffer transfer only	Side 1	ϕ	ϕ	WRITE	ϕ

The read/write instruction must be followed by
INQUIRE DISK or QUERY DISK FORMAT

Query Instruction

"Uϕ" CHR$ (control) CHR$ (offset track)

	BIT		7	6	5	4	3	2	1	ϕ
Control	INQUIRE DISK	ϕ	–	–	–	Side ϕ	ϕ	1	ϕ	ϕ
		1	–	–	–	Side 1 (MFM)				
	QUERY DISK FORMAT	ϕ	–	–	–	Side ϕ	1	ϕ	1	ϕ
		1	Offset track	–	–	Side 1				
Offset Track			Active if bit 7 of QUERY DISK FORMAT control byte is set.							

The query instruction should be send through direct command.

Sector interleave Instruction

"Uϕ" CHR$ (control) CHR$ (interleave)

Control	$88	Brust handshake
	$ϕ8	Write

Utility instruction

"Uϕ>S" CHR$ (sector interleave) : REM DOS sector interleave
"Uϕ>R" CHR$ (retries) : REM DOS retrives
"Uϕ>T" : REM Run diagnostic
"Uϕ>M1": REM 1571 Mode
"Uϕ>Mϕ": REM 1541 Mode
"Uϕ>Hϕ": REM 1541 Head zero
"Uϕ>H1": REM 1541 Head one
"Uϕ>" CHR$ (device) : REM device no. change 4 - 3ϕ

Fast load instruction

"U$" CHR$ (control) "filename $" :

control byte	CHR$ (31) for Program file
	CHR$ (255) for Sequential file

	$ϕϕ or $ϕ1	OK
Echo Message	$ϕ2	File not found
	$1F	EOI

Status Byte

FORMAT	7	6	5 4	3	2	1	ϕ	MEANING
GCR	ϕ	ϕ		0	0	0	0	OK
				0	0	0	1	
	ϕ	ϕ		0	0	1	0	Sector Not Found
				0	0	1	1	No SYNC
	ϕ	ϕ		0	1	0	0	Data Block not Found
				0	1	0	1	Data Block Checksum ERROR
	ϕ	ϕ		0	1	1	0	Format Error
				0	1	1	1	Verify Error
	ϕ	ϕ		1	0	0	0	Write Protect Error
				1	0	0	1	Meader Block Checksum Error
	ϕ	ϕ		1	0	1	0	Data Extended into Next Block
				1	0	1	1	Disk ID Mismatch
	ϕ	ϕ	00-128 byte/sector	1	1	0	0	Reserved
				1	1	0	1	
	ϕ	ϕ	01-256 byte/sector	1	1	1	0	SYNTAX Error
				1	1	1	1	No drive Present
MFM	1	ϕ	10-512 byte/sector	0	0	0	0	OK
				0	0	0	1	
	1	ϕ	11-1024 byte/sector	0	0	1	0	Sector Not Found
				0	0	1	1	No Address Mark
	1	ϕ	(MFM)	0	1	0	0	Reserved
				0	1	0	1	Data CRC Error
	1	ϕ	only	0	1	1	0	Format Error
				0	1	1	1	Verify Error
	1	ϕ		1	0	0	0	Write Protect Error
				1	0	0	1	Header Block Checksum Error
	1	ϕ		1	0	1	0	Reserved
				1	0	1	1	Disk Change
	1	ϕ		1	1	0	0	Reserved
				1	1	0	1	
	1	ϕ		1	1	1	0	SYNTAX Error
				1	1	1	1	No Drive Present

7.6 OTHER USER COMMANDS

FORMAT PRINT#file#,"U command #"

PURPOSE To jump to a particular location in the buffer or memory for execution of different programs.

DESCRIPTION In addition to the USER1 and USER2 commands discussed before, the USER commands listed in the table below cause a program to jump to specific locations in the buffer memory to execute different programs that may have been loaded into the memory.

U COMMAND	DESCRIPTION	EXECUTION
U1 or UA	read first byte of the block and funtcion as BLOCK-READ	CD5F
U2 or UB	set buffer pointer to one and function as BLOCK-WRITE	CD97
U3 pr UC	user define	0500
U4 or UD	user define	0503
U5 or UE	user fefine	0506
U6 or UF	user define	0509
U7 or UG	user define	050C
U8 or UH	user define	050F
U9 or UI	check U1+, UI-	FF01
U: or UJ	reset	EAA0
UI+	confirm Commodore 64 speed	(0065)
UI–	confirm VIC-20 speed	FF0D

EXAMPLE
```
PRINT#15,"UJ":FOR I=1 TO 1000:NEXT I
PRINT#15 "U1 2 0 18, 1"
PRINT#15,"UI+"
```

CHAPTER 8 CHANGE OF DEVICE NUMBER

The drive comes with the factory selected device number 8 and the drive number 0. This is the usual device number, but if more than one disk drive is to be used with the system, it is necessary to give each disk drive a different device number. It is impossible to change the drive number but the device number can be changed using either software or hardware method.

8.1 SOFTWARE METHOD

The device number is changed by performing a MEMORY-WRITE to locations $0077 and $0078. The command is executed once the command channel has been opened.

FORMAT "Uϕ>"CHR$ (New device adderss)

EXAMPLE The following routine changes the device number to 9 .

```
10 OPEN 15,8,15
20 PRINT#15,"Uϕ>" CHR$(9)
```

It is usually desirable to change the device number in hardware unless a temporary change is all that is desired. In order to use the software method, only one drive can be powered on, its device number changed, then another drive is powered on and its device number changed until all drives are on. IF THIS PROCEDURE IS NOT FOLLOWED, THE DISK DRIVES' DEVICE NUMBERS WILL CONFLICT WITH ONE ANOTHER.

8.2 DIP SWITCHES SETTING

On the BCD 128 disk drive, there are two DIP switches on the back of unit which determine the drive's device number. You can defince the device number by changing the setting of the switches with a small flat screwdriver.

The following table lists the switch settings for changing the device numbers:

DEVICE # SELECTION	SWITCH			
	1	2	3	4
8	ON	ON	ON	ON
9	OFF	OFF	ON	ON
10	ON	ON	OFF	OFF
11	OFF	OFF	OFF	OFF

APPENDIX A DISK DRIVE ERROR MESSAGES

These error messages generated by the disk drive are useful for identifing problems with the media or data when you encounter difficulty in reading or writing data. Given below is a full list of the error messages which might be displayed.

00: OK (not an error)
This is the message that usually appears when the error channel is checked. It means there is no current error in the disk unit.

01: FILES SCRATCHED (not an error)
This is the message that appears when the error channel is checked after using the SCRATCH command. The track number tells how many files were erased.

NOTE: If any other error message numbers less than 20 ever appear, they may be ignored. All true errors have numbers of 20 or more.

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 block or a header that has been destroyed. Usually unrecoverable.

21: READ ERROR (no sync character)
The disk controller is unable to detect a sync mark on the desired track. Caused by misalignment, or a diskette that is absent, unformatted or improperly seated. Can also indicate hardware failure. Unless caused by one of the above simple causes, this error is usually unrecoverable.

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. Occurs in conjunction with BLOCK commands and indicates an illegal track and/or sector request.

23: READ ERROR (checksum error in data block)
There is an error in the data. The sector has been read into disk memory, but its checksum is wrong. May indicate grounding problems. This fairly minor error is often repairable by reading simply and rewriting the sector with direct access commands.

24: READ ERROR (byte decoding error)
The data or header has been read into disk memory, but a hardware error has been created by an invalid bit pattern in the data byte. May indicate grounding problems.

25: WRITE ERROR (write-verify error)
The controller has detected a mismatch between the data written to diskette and the same data in disk memory. May mean the diskette is faulty. If so, try another. Use only high-quality diskettes from reputable makers.

26: WRITE PROTECT ON
The controller has been requested to write a data block while the write-protect sensor is covered. Usually caused by writing to a diskette whose write protect notch is covered over with tape to prevent changing the diskette's contents.

27: READ ERROR (checksum error in header)
The controller detected an error in the header bytes of the requested data block. The block was not read into disk memory. May indicate grounding problems. Usually unrecoverable.

28: WRITE ERROR (long data block)
The controller attempts to detect the sync mark of the next header after writing a data block. If the sync mark does not appear on time, this error message is generated. It is caused by a bad diskette format (the data extends into the next block) or by a hardware failure.

29: DISK ID MISMATCH
The disk controller has been requested to access a diskette which has not been initialized. Can also occur if a diskette has a bad header.

30: SYNTAX ERROR (general syntax)
The DOS cannot interpret the command sent to the command channel. Typically, this is caused by an illegal number of file names or an illegal pattern. Check your typing and try again.

31: SYNTAX ERROR (invalid command)
The DOS does not recognize the command. It must begin with the first character sent. Check your typing and try again.

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

33: SYNTAX ERROR (invalid file name)
Pattern matching characters cannot be used in the SAVE command or when Opening files for the purpose of Writing new data. Spell out the file name.

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 omitted. Try again.

39: SYNTAX ERROR (invalid command)
The DOS does not recognize a command sent to the command channel (secondary address 15). Check your typing and try again.

50: RECORD NOT PRESENT
The requested record number has not been created yet. This is not an error in a new relative file or one that is being intentionally expanded. It results from reading past the last existing record, or positioning to a non-existent record number with the Record# command.

51: OVERFLOW IN RECORD
The data to be written in the current record exceeds the record size. The excess has been truncated (cut off). Be sure to include all special characters (such as carriage returns) in calculating record sizes.

52: FILE TOO LARGE
There isn't room left on the diskette to create the requested relative record. To avoid this error, create the last record number that will be needed as you first create the file. If the file is too large for the diskette, either split it into two files on two diskettes, or use abbreviations in the data to allow shorter records.

60: WRITE FILE OPEN
A write file that has not been closed is being reopened for reading. This file must be immediately rescued, as described in BASIC Hint #2 in Chapter 2, or it will become a splat (improperly closed) file and probably be lost.

61: FILE NOT OPEN
A file is being accessed that has not been opened by the DOS. In some such cases no error message is generated. Rather the request is simply ignored.

62: FILE NOT FOUND
The requested file does not exist on the indicated drive. Check your spelling and try again.

63: FILE EXISTS
A file with the same name as has been requested for a new file already exists on the diskette. Duplicate file names are not allowed. Select another name.

64: FILE TYPE MISMATCH
The requested file access is not possible using files of the type named. Reread the chapter covering that file type.

65: NO BLOCK
Occurs in conjunction with B-A. The sector you tried to allocate is already allocated. The track and sector numbers returned are the next highter track

and sector available. If the track number returned is ϕ, all remiaining sectors are full. If the diskette is not full yet, try a lower track and sector.

66: ILLEGAL TRACK AND SECTOR
The DOS has attempted to access a track or sector which does not exist. May indicate a faulty link pointer in a data block.

67: ILLEGAL SYSTEM T OR S
This special error message indicates an illegal system track or block.

70: NO CHANNEL (available)
The requested channel is not available or all channels are in use. A maximum of three sequential files or one relative file plus one sequential file may be opened at one time, plus the command channel. Do not omit the drive number in a sequential OPEN command, or only two sequential files can be used. Close all files as soon as you no longer need them.

71: DIRECTORY ERROR
The BAM (Block Availability Map) on the diskette does not match the copy in disk memory. To correct, Initialize the diskette.

72: DISK FULL
Either the diskette or its directory is full. DISK FULL is sent when two blocks are still available, allowing the current file to be closed. If you get this message and the directory shows any blocks left, you have too many separate files in your directory, and will need to combine some, delete any that are no longer needed, or copy some to another diskette.

73: DOS MISMATCH (CBM DOS V3.0.1571)
If the disk-error status is checked when the drive is first turned on, before a directory or other command has been given, this message will appear. In that use, it is not an error, but rather an easy way to see which version of DOS is in use. If the message appears at other times, an attempt has been made to write to a diskette with an incompatible format, such as the former DOS 1 on the Commodore 2040 disk drive. Use one of the copy programs on the Test/Demo diskette to copy the desired file(s) to a 1571 diskette.

74: DRIVE NOT READY
An attmept has been made to access the 1571 single disk without a formatted diskette in place. Blank diskettes cannot be used until they have been formatted.

APPENDIX B DISK FORMAT

1. RELATIVE FILE FORMAT

see chapter 6.1

2. SEQUENTIAL FILE FORMAT

BYTE	MEANING
*0,1	Track and block of next block.
2-255	254 bytes of data with carriage return as terminators.

3. PROGRAM FILE FORMAT

BYTE	MEANING
*0,1	Track and block of next block.
2-255	CBM memory format with 254 bytes of program data stored. Byte 2, 3 of the first block represent the start location in the computer when program is loaded. Three zero bytes are used for EOF (end of file).

*Note: Byte 0 = zero means this is the last block and Byte 1 is the no. of used bytes in this block.

4. DISKETTE SECTORS & TRACKS ASSIGNMENT

TRACK	BLOCK	BIT TIME	SECTORS/TRACK
01-17	0-20	3.25 μsec	21
18-24	0-18	3.50 μsec	19
25-30	0-17	3.75 μsec	18
31-35	0-16	4.00 μsec	17

5. HEADER OF DIRECTORY

Track 18	Sector 0	
BYTE	CONTENT	MEANING
0	18 ($12)	Track of first directory block
1	1 ($ϕ1)	Sector of first directory block
2	65 ($41)	"A"
	0 ($ϕϕ)	Signle-side flag
3	128 ($8ϕ)	Double-side flag
4*TRK#+ϕ		Number of available block on the track
4*TRK#+1	BAM	Sector 0 – 7 BAM of the track
4*TRK#+2	of	Sector 8 – 16 BAM of the track
4*TRK#+3	Track 1 – 35	Sector 17 – 23 BAM of the track
144 – 159		Disk name (filled with "Shift space")
16ϕ, 161	16ϕ, 16ϕ ($Aϕ, Aϕ)	Shift Spaces
162, 163		ID Mark
164	160	Shift spaces
165, 166	50, 65 ($32, 41)	"2A"
167 – 17ϕ	160,....($Aϕ....)	Shift spaces
171 – 179	ϕ	Filled with zeros
180 – 191		BLOCKS FREE.
192 – 22ϕ	ϕ	Filled with zeros
TRK#+185	For track 36 – 70	Number of available block on the track

Note: Above numbers represents a BIT MAP of the tracks. Each bit no. represents a specified block.
Flag of block: 1–available
0–unavailable

6. BAM on Side One (1571 Mode)

TRACK 53	SECTOR ϕ	
BYTE	CONTENTS	MEANING
3*TRK–1ϕ8	BAM	Sector 0 – 7 BAM of the track
3*TRK– ϕ7	of	Sector 8 – 16 BAM of hte track
3*TRK– ϕ6	Track 36 – 70	Sector 17 – 23 BAM of the track
1ϕ5-255	ϕ	Filled with zero

Note: Above numbers represents a BIT MAP of the tracks. Each bit no. represents a specified block.
Flag of block: 1–available
0–unavailable

7. DIRECTORY DESCRIPTION

<table>
<tr><td colspan="3">TRACK 18, SECTOR 1</td></tr>
<tr><td>BYTE</td><td colspan="2">MEANING</td></tr>
<tr><td>0,1</td><td colspan="2">Track and block of next block</td></tr>
<tr><td rowspan="5">30*file entry no.+2</td><td>Bit</td><td>Description</td></tr>
<tr><td>0-3</td><td>File type
0-DELeted
1-SEQuential
2-PRoGram
3-USeR
4-RELative</td></tr>
<tr><td>4,5</td><td>Don't care</td></tr>
<tr><td>6</td><td>Lock Flag
0-unlocked file
1-locked file</td></tr>
<tr><td>7</td><td>File Flag
0-file opened
1-file closed</td></tr>
<tr><td>30*file entry no.+3,4</td><td colspan="2">Track and block of the first data block</td></tr>
<tr><td>30*file entry no.+5-20</td><td colspan="2">File name (filled with shifted space)</td></tr>
<tr><td>30*file entry no.+21,22</td><td colspan="2">Track and block of the first block of replacement file when overwritten with @ :</td></tr>
<tr><td>30*file entry no.+23-27</td><td colspan="2">record lenght (relative file only)</td></tr>
<tr><td>30*file entry no.+24-27</td><td colspan="2">Don't care</td></tr>
<tr><td>30*file entry no.+28,29</td><td colspan="2">Track and block of the first block of replacement file when overwritten with @ :</td></tr>
<tr><td>30*file entry no.+30,31</td><td colspan="2">low byte and high byte of the no. of block in file</td></tr>
</table>

APPENDIX C SERIAL INTERFACE SIGNALS

BCD/5.25 disk drive is capable of daisy-chaining up to 4 drives using serial communication.

The pin assignment for the serial bus is as follows:

PIN	SIGNAL	DESCRIPTION
1	SRQ IN	Used by fast serial bus as a bi-direction fast clock line. Unused by the slow serial bus.
2	GND	Logic ground
+	ATN I/O	SERIAL ATTENTION IN/OUT The computer use this channel to start a command sequence.
4	CLK I/O & DATA I/O	SERIAL CLOCK IN/OUT and SERIAL DATA IN/OUT These lines are used for DATA transfer.
6	RESET	RESET When this line is set LOW by the host system, the CPU of the device will be reset and all registers will assume the default value.

PIN NO.	SIGNAL
1	SRQ IN
2	GND
3	ATN I/O
4	CLK I/O
5	DATA I/O
6	RESET

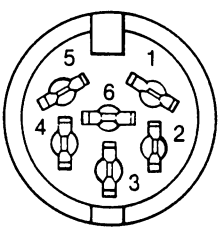


Figure 7 Serial I/O Connector

APPENDIX D SPECIFICATIONS

- Interface: Commodore compatible, dual serial port with daisychain option.
- Number of Tracks: 70
- Number of Heads: Two
- Data Encoding Method: Group Coded Recording (GCR)
- Net Weight: 3 lbs. 12 0z.
- Capacity:

	Unformatted	Formatted
GCR	504K	350K
MFM	500K	265K–405K

- Buffer Memory: 2048 bytes
- Track Density: 48 Tracks per inch.
- Directory Entries: 144
- Sectors per Side: 17 to 21
- Bytes per Sector: 256
- Blocks per Side: 1366
- Internal Write Protection: Yes
- Power-On Diagnostics: Yes
- Power Consumption: 15 Watts (Typical)
- Power Supply: External – 16 VAC, 0.8 Amps.
  9 VAC, 1.5 Amps.
- Power Requirements: 120 VAC, 60 Hz.
- Dimensions: 3.0" X 6.75" X 10.6"

- Reliability: Mean Time Between Failure (MTBF)– 10,000 Power-on hours.
- Media Requirements: Industry Standard (ANSI) 4¼" diskettes; soft secotred.
- Compatible Computers: Commodore 64, Commodore 128, SX64, PLUS4, C16, and VIC 20.

BLUE CHIP ELECTRONICS, INC.

Blue Chip Electronics, 7305 W. Boston Street, Chander, Arizona 85226