65D TUTORIAL and REFERENCE MANUAL

A Step by Step Introduction to the Ohio Scientific OS-65D Operating System

August, 1981

A T T E N T I O N

This set contains six disks: The five disk Tutorial Set and one blank disk. The tutorial disks have blue labels and the blank disk has a brown label.

T A B L E O F C O N T E N T S

TABLE OF CONTENTS (Cont'd)

T A B L E O F C O N T E N T S (Cont'd)

Introduction

This manual consists of two parts. Part one (chapters 1 through 5) is a tutorial introduction to the wide range of features and utilities afforded the user by Ohio Scientific's OS-65D operating system. Because it is a tutorial, part one should be studied thoroughly, following the proper sequence of chapters. The reader who jumps ahead before mastering the early chapters risks possible confusion.

Each chapter of the tutorial makes use of one of the five Tutorial Disks that accompany this manual. The early Tutorial Disks have been specially designed to make it possible for the reader to write and use relatively sophisticated programs, that process disk data files, without first learning all of the complicated details of file manipulation. Hence, mastery of some of the more difficult technicalities of the operating system can be delayed until their necessity is properly motivated.

Part two of this manual (beginning with chapter 6) provides complete information on all of the versions of OS-65D, including the new Version 3.3. This material is intended to be a natural continuation for the beginner who has worked through the part one tutorial. It can also be used as a stand-alone reference for the experienced user who is familiar with OS-65D version 3.2.

It is assumed that all users of this manual have a working (not detailed!) knowledge of the computer language BASIC. If you do not have this background, please acquire a copy of OSI's BASIC and the Personal Computer (standard with some OSI computers) and read Chapter 2. Any other standard text on introductory BASIC will also suffice. Tutorial Disk 1 should prove to be a useful tool to anyone learning BASIC for the first time or practicing some old skills.

If you are familiar with BASIC on another computer system, it is suggested that you read through the OSI BASIC Reference Manual for detailed examples of OSI BASIC. This is part of the system documentation for nearly all OSI computers.

A brief description of the five Tutorial Disks:

Disk 1 This contains a collection of interactive programs. It is designed to be an easy-to-use, entertaining and informative introduction to the system.

Disk 2 The user is introduced to the BASIC workspace, enters a sample program, and learns how to correct typing errors.

Disk 3 This disk contains eight pre-created files for BASIC programs. The user learns how to store and retrieve BASIC programs from the disk. File names are introduced.

Disk 4 This disk comes with three files for BASIC programs and four data files. Sample programs illustrate sequential file usage, random file usage, and combinations of the two. The concept of a "buffer" is introduced.

Disk 5 This disk is a full OS-65D V3.3 system disk. The directory is introduced. Full file create, delete and rename utilities are examined. Disk copying on single and dual drive machines is explained. The 65D operating system is discussed.

As the above descriptions indicate, the disks are numbered in order of increasing sophistication. Each disk is used with the correspondingly numbered chapter in the tutorial.

Users who are thoroughly familiar with OS-65D V3.2 may want to proceed directly to Chapter 7 for an overview of the many new features of version 3.3.

IMPORTANT NOTE TO USERS OF SERIAL SYSTEMS (C-2,C-3 SYSTEMS)

Four of the "games" programs contained on Tutorial Disk 1 are operable only on video systems (C1P, C4P, C8P systems). If you select any of these programs on a serial system they will "hang-up" the computer, forcing you to reboot. The Hangman, Biorhythm and Loan Interest programs will, however, run properly on a serial system.

Certain new features of the OS-65D V3.3 operating system do not work on serial systems. Generally, commands that refer to color, screen size, or windowing are not operable. A complete list of the non-working commands is provided in Chapter 7 (see page 100).

If your system includes a Hazeltine 1420 terminal, be sure that switch 6 is set to the ESC position (on).

IMPORTANT NOTE TO USERS OF C1P SYSTEMS

Your computer system must have a minimum of 24K of memory to properly run the OS-65D Tutorial Disks. A C1P with only 20K of memory will not boot up with Tutorial Disk 5 because its menu program (BEXEC*) will not fit into the 20K workspace.

It is, however, possible to use other Tutorial Disks with a 20K C1P. For example, Tutorial Disks 3 and 4 could be used to boot the system so that the C1P user could use the editor and other new features of OS-65D V3.3.

TO START YOUR COMPUTER

1. Turn on the computer, disk drives and terminals - switches are generally located on the back of the device cabinet.

2. Place an OS-65D disk in drive A (the drive whose red light is on or the top drive in dual drive cabinets). Close the disk drive door.

3. Depress the BREAK key on C1P and C4P systems (and hold for a few seconds). Depress the white reset button on C8P and serial systems.

4. When the "H/D/M?" ("D/C/W/M?" on C1P systems) message appears, respond by typing "D" and <RETURN>. In a few seconds a menu should appear on the screen.

5. To enter the BASIC immediate mode, respond UNLOCK to this menu in OS-65D V3.2; select option 9 in OS-65D V3.3.

Chapter 1

Tutorial Disk One

This disk contains seven programs which can be run by a user with no programming knowledge.

The procedure for using Tutorial Disk 1 follows.

A) After the computer, disk drive(s), and monitor have been connected and turned on (according to the instructions in your user's manual) depress <BREAK> (or the white reset button on your computer) and keep it depressed for about two seconds or until 'H/D/M' ('D/C/W/M' on C1P systems) appears at the top of the screen. Then insert Tutorial Disk 1 into the disk drive, (label up and toward the user) and close the door of the drive. If you have a system with more than one drive, insert the disk into drive A.The red light on drive A will be lit after depressing <BREAK> on minifloppy systems; if you have dual disk drives in a single cabinet, drive A is the top drive.

B) Make sure the SHIFT-LOCK (or ALL-CAPS) key is depressed.

C) Depress D (for disk).

After a short delay, the menu below will appear on the screen.

```
OS-65D Tutorial disk one

  1> Space War
  2> Hangman
  3> Biorhythm
  4> Torpedo
  5> Breakout
  6> Loan Interest Calculation
  7> Life

Depress the number of your selection ?
```

When the menu is on the screen, make a selection and type its number (1-7). Selections 1,2,4 and 5 are one and two player games. Selections 3 and 6 prompt the user for certain information and then display a report. Finally, selection 7 is a graphics demonstration program. It is not available on C1P systems.

As was mentioned in the introduction, selections 1,4 and 5 will run only on video systems (C1P, C4P and C8P). Selection 7 also runs only on video systems; it is particularly impressive on color video systems. Selections 2, 3 and 6 will run on all OSI computer systems, including serial systems. The following is a brief description of each of these games.

1> Space wars is a two player game of skill where the ground forces battle the starship Enterprise in an intergalactic battle.

2> Hangman is a word guessing game. Using the positions of letters that you have correctly selected as clues, you must guess the computer's word before you are "hanged".

3> Biorhythm will plot your emotional, physical and intellectual cycles for a 30-day period. It will display high, low and "critical" days. The program uses the popular (and controversial) Biorhythm theory for its calculations.

4> Torpedo is a one player game where you are a commander, and must destroy all enemy boats that enter your waters.

5> Breakout is a one player game where you try to keep the ball moving around the screen and knock out as many blocks as you can to score points.

6> Loan Interest Calculation will display a repayment schedule and total interest charges for any loan. You select the principal amount, interest rate, time period, and either the number of payments or amount of each payment.

7> Life is an animated version of the classic simulation of population growth and decay. The small squares in the display are "born" when they have the optimal number of "neighbors". Too few or too many neighbors causes a square to die and disappear from the display. The entire population will eventually reach a stable state or be completely wiped out. Life was invented by mathematician George Conway. It has been extensively discussed in Scientific American.

When one of these programs is selected, the computer first displays a discussion or directions. Take as long as you wish to read the display and then respond to the computer's prompt for input from the user by making an appropriate entry on the keyboard. In the case of programs 2, 3 and 6, it will be necessary to press the RETURN or CR key after typing certain kinds of information in order to complete the entry. For the other programs, the RETURN or CR key is never used. You should be certain that the SHIFT-LOCK (or ALL-CAPS) key is down while using the programs on Tutorial Disk 1.

The user may break out of any one of programs 1, 2, 4, 5 or 7 by depressing <ESC> (escape key). It may be necessary to hold <ESC> down for a second or two before the computer responds. When it does it will return to the menu. When any of the programs has finished running (without interruption through the use of <ESC>), the computer asks if the user wants to run it again. Simply pressing the Y key (for yes) or the N key (for no) will rerun the program or return to the menu.

The user may end the use of this disk at any time by depressing <BREAK> (or the white reset button) and removing the disk from the disk drive. Any disk in a disk drive should always be removed before turning off the disk drive.

Chapter 2

Tutorial Disk Two

Tutorial Disk 2 allows the user to enter his own BASIC commands or BASIC programs. There is, however, no means for saving a program when using this disk. If the user wants to enter a program and save it on the disk, he should use Tutorial Disk 3. We will use the following terminology:

boot-up: The process of initializing the computer and loading software from the disk that permits the user to enter BASIC commands or lines of a program. Boot-up is accomplished by depressing <BREAK> on the keyboard (C1P and C4P systems) or the white reset button on the computer (C8P and serial systems) until the 'H/D/M?' ('D/C/W/M?' on C1P systems) message appears on the screen. Respond by typing D (for disk) with the SHIFT-LOCK (ALL CAPS) key down.

workspace: The part of memory in which the user's program is stored. This is RAM or random access memory.

prompt: The symbol or message that is displayed when the computer is waiting for a command to be entered from the keyboard. If the computer is waiting for a BASIC command or program entry then the "OK" prompt is displayed. If a BASIC program is running (e.g., the game programs on Tutorial Disk 1) and an INPUT statement is encountered then the "?" prompt is displayed. Other prompts will be discussed in conjunction with other disks.

When Tutorial Disk 2 is booted-up, the workspace is cleared of all previous material and the computer displays the OK prompt. When other disks are booted-up, the workspace may not be cleared. At this point the user may enter BASIC commands either without a line number for immediate execution or with a line number to be stored in the workspace as part of the user's program. (The former method is called the BASIC Immediate Mode.) We will give a brief discussion about BASIC in the following sections. For a more in-depth discussion about BASIC see Dwyer and Critchfield's book BASIC and The Personal Computer or the OSI BASIC Reference Manual.

A) The BASIC Immediate Mode

When the user enters (types) a BASIC command and the typed line does not begin with a line number, then the command is executed immediately. When the OK prompt is displayed each line

that is entered from the keyboard must be terminated by depressing <RETURN> (<CR>). Most BASIC commands can be entered in the immediate mode, that is, without a line number. The exceptions are DEF, INPUT, READ, and DATA. Commands like GOTO that must reference a line number can be entered in the immediate mode only if there is a program in the workspace to reference.

One way to use the computer in the immediate mode is as a calculator. This usually involves the BASIC "PRINT" command which causes the computer to display a result. In OSI's Microsoft BASIC, a question mark (?) can be used as an abbreviation for the PRINT command.

Some of the arithmetic symbols used by OSI's BASIC are:

```
+  add
-  subtract
*  multiply
/  divide
   (or SHIFT-N)  exponentiation
SQR  square root
SIN  sine of a number
```

For example, if the OK prompt is on the screen and the user enters the command

```
? SQR (2)  or  PRINT SQR (2)
```

(followed by <RETURN>)

then the display will be:

```
OK
? SQR (2)
 1.41421356

OK
```

That is, the computer will display (or PRINT) the square root of 2. (NOTE: A typing error can be corrected by pressing <RUB-OUT> or <SHIFT-O>.) If the user enters the command

```
? 3*(61.8Ø9-12.929)
```

then the computer will display the result of that calculation, as follows:

```
OK
? 3*(61.8Ø9-12.929)
 146.64

OK
```

(Note, the zeros have a slash through them (Ø) to differentiate them from the fifteenth letter of the alphabet.)

The user can cause values to be stored in memory. For example, if the user enters

```
PI=3.14159
```

and then

```
? PI*(4^2)
```

then the display will be:

```
OK
PI=3.14159

OK
? PI*(4^2)
 50.26544

OK
```

That is, the computer will print the area of the circle of radius 4. The value stored at PI will be available for later calculations. So, for example, if the user now enters the command

```
? PI*(17.3^2)
```

then the display would be:

```
OK
? PI*(17.3^2)
 940.246472

OK
```

More than one command can be entered at one time, provided the commands are separated by colons. For example, the value of the formula

```
X^4 - 4*X^3 + 3*X + 9
```

for X=7

can be displayed as follows:

```
OK
X=7: ? X^4 - 4*X^3 + 3*X + 9
 1059

OK
```

(Remember that the "^" symbol is typed by pressing <SHIFT-N> on some keyboards.)

The FOR-NEXT pair can be used in the immediate mode, but it must be on one line. For example, a table of square roots could be

obtained as shown below:

```
OK
FOR X=1 TO 10: ?X,SQR(X): NEXT X
 1             1
 2             1.41421356
 3             1.73205081
 4             2
 5             2.23606798
 6             2.44948974
 7             2.64575131
 8             2.82842713
 9             3
 10            3.16227766

OK
```

The PEEK and POKE commands are frequently used in the immediate mode. For example, the command

```
POKE 56832,0
```

will cause the C4P and C8P monitor screen displays to go into the 32x32 mode (32 lines of 32 characters) with color and sound generators turned off. Entering the command

```
POKE 56832,1
```

will return the display to its original 32x64 mode (32 lines of 64 characters) with sound and color turned off. See your user's manual for more information on memory location 56832.

B) Entering a BASIC Program

When the OK prompt is on the screen the user may enter lines of BASIC code to be stored in the workspace for later execution. Each such line must begin with a line number. Remember, there is no provision on Tutorial Disk 2 for storing programs. If the user wants to store a program on a disk then Tutorial Disk 3 may be used. As a short example, the user might type in:

```
10 PRINT"TEST PROGRAM"
20 PRINT
30 INPUT A$
40 PRINT LEN(A$)
50 IF A$ = "DONE" THEN STOP
60 GOTO 20
```

If you make a mistake (before you depress <RETURN>) use the SHIFT-O, SHIFT-"underline" (next to the 0 key), or RUB-OUT key to delete the previous character. (Different keys implement the delete function on different systems).

If you notice a typing error in a line that has already been typed, then type the line again with the same line number. The new version will replace the old. Typing a line number followed immediately by the depression of the <RETURN> key, will delete

the line with that number. To display the program as it currently is in memory, type the command

```
LIST
```

At times you may want to list only part of a program. You may use the LIST command followed by a line number or numbers. Examples are:

```
LIST 20       list just line 20
LIST 20-40    list lines 20 through 40 inclusive
LIST -40      list from the first line through line 40
LIST 20-      list from line 20 to the end of the program
```

The program, when correctly entered will:

1) print the title "TEST PROGRAM"
2) ask for information to be typed in at the keyboard
3) count and display the number of characters that were entered
4) stop if DONE was entered, otherwise go back to step 2 and repeat the process.

To run this program type the command

```
RUN
```

The computer should display

```
TEST PROGRAM
```

and then

```
?
```

which is a prompt, indicating that the computer is waiting for information from the keyboard and a carriage return (depression of the <RETURN> key). Anything may be typed at this point except <BREAK>. If, at any point, an error message is displayed, then the program was not typed in correctly, and the programmer should LIST it, correct any errors and then RUN the program again. If the program has no errors and is running, it may be stopped by responding with DONE to the ? prompt.

Chapter 3

Tutorial Disk Three

Tutorial Disk 3 is similar to Tutorial Disk 2 in that the user can enter commands in the immediate mode (Chapter 2) and type in BASIC programs. However, Tutorial Disk 3 also allows for storing programs on disk.

For purposes of disk storage, a BASIC program or collection of data records is called a file. Thus we may refer to a program file or data file. Data files are discussed in detail in Chapter 4.

When the programmer wishes to store a program on disk there must already be a place (file) on the disk to store it. Tutorial Disk 3 contains eight precreated files that are ready for immediate storage of programs. The techniques for creating and manipulating disk files are discussed in detail in Chapter 5. When a program file is brought from disk into the workspace, we say the file is loaded from the disk. When this happens, the file on disk remains unchanged while being copied into memory. The computer reads the disk. When you read a book the information remains on the pages of the book. Likewise, when the computer reads the disk that file remains on disk. After the file has been loaded into the workspace, the programmer may then LIST it, RUN it or make changes and then save the updated version on disk, either in place of the old versions or in another file so that both versions are on disk simultaneously.

When Tutorial Disk 3 is booted-up, the computer displays the following menu and waits for the user to type a menu number (as with Tutorial Disk 1 except now you must also depress <RETURN>).

```
        1 >  LOAD program called "PROG1"

        2 >  LOAD program called "PROG2"

        3 >  LOAD program called "PROG3"

        4 >  LOAD program called "PROG4"

        5 >  LOAD program called "PROG5"

        6 >  LOAD program called "PROG6"

        7 >  LOAD program called "PROG7"

        8 >  LOAD program called "PROG8"

        9 >  Rename a program file

Type the number of your selection and depress <RETURN>
```

The menu gives a list of the eight precreated files and, in addition, offers a ninth option for renaming files. If the programmer enters 1 through 8 then the computer will load the specified program into the workspace, ready to LIST, RUN or modify. Initially, each file contains a short filler program. To replace a filler program with one which you have written, use the following procedure:

1) Select one of the eight files by typing the number corresponding to your choice, then press <RETURN>.

2) Type

   NEW<RETURN>

   to clear the workspace and erase the filler program from the workspace (the copy of the program on disk is unchanged).

3) Enter your program as in section B of Chapter 2.

4) Type

   DISK!"PUT PROG1"<RETURN>

   If you have already used option 9 to rename PROG1 then you should use the new name instead of PROG1. You can save your program in any file that is listed in the menu by using its name in place of PROG1.

Below is a complete example. The example program, when RUN, displays all 255 graphics characters on the screen. For more information on graphics see your user's manual. The first line scrolls the screen display upward, clearing the screen so that the output of this program can be viewed without interference from the previous screen display. The second line changes the screen display to the 32x32 mode on a C4P or C8P computer so that the displayed characters will be larger. When the program has stopped running the screen will remain in 32x32 mode. To return to the standard 32x64 mode the programmer can type

```
POKE 56832,1
```

The listing at the top of the next page is exactly as it would appear on the screen, including the OK prompts. Everything except the OK prompts was entered by the programmer. Use <RUB-OUT> to correct any typing errors (see Chapter 2). It is assumed that the computer has been booted-up and the user has selected option 1.

```
OK
NEW

OK
 90 FOR S=1 TO 25:PRINT:NEXT
100 POKE 56832,0
200 C=0
300 FOR L=0 TO 15
400 FOR P=0 TO 30 STEP 2
500 POKE 53760+64*L+P,C
600 C=C+1
700 NEXT P
800 NEXT L

DISK!"PUT PROG1"

OK
```

Now that the program is saved on disk, you may want to run the program by typing RUN.

To run the program on a C1P system, the following changes are necessary:

```
300 FOR L=0 TO 7
400 FOR P=0 TO 31
500 POKE 53586+64*L+P+7,C
```

The program cannot be modified to run on serial systems because it uses the memory-mapped screen display features of the OSI video systems.

When the menu is on the screen, the user can change the name of any of the eight files by selecting option 9. When this option is selected, the computer will ask for the old file name (that is the name before any change is made) and then ask for the new name. The names can be up to six characters long, and the first character must be a letter. Examples:

Valid names	Invalid names
PROG1	1PROG
TESTIT	TEST IT
SAMPLE	*FILE

Only the name of the file will be changed; the program stored in the file will not be affected. When the user stores his own program in one of the eight files it may be desirable to give it a more descriptive name. This is, however, completely optional.

Any file that exists on a disk has a specific length, given as a number of tracks. One disk track can hold slightly more than 2000 bytes of data or 2000 characters of a BASIC program (3000 bytes of data or characters on eight inch disks). Each of the precreated files on Tutorial Disk 3 has a length of three

tracks. Hence, a longer program cannot be stored on Tutorial Disk 3. The creation of files of user-specified lengths is discussed in Chapter 5.

The programmer can easily determine the length, in tracks, of the program currently in the workspace. The procedure is:

1) Following the OK prompt, type

   EXIT<RETURN>

   The computer will respond with the length of the program and then display an A* prompt. The A* prompt means that the computer is no longer in the BASIC mode but in the Disk Operating System (DOS) mode. This is discussed further in Chapter Six.

2) To return to BASIC from DOS type:

   A*RETURN BASIC<RETURN> or A*RE BA<RETURN>

The program that is RUN upon boot-up is called MENU. The name MENU was used for simplicity. Later you will find that the program called BEXEC* is the program RUN on boot-up for most diskettes. MENU is the program that prints the menu and processes your selection. The line

RUN"MENU<RETURN>

may be put into any program or executed in the immediate mode. When this command is executed, the program MENU will be loaded into the workspace (hence erasing any other program in the workspace) and the menu will be displayed.

Chapter 4

Tutorial Disk Four

Tutorial Disk 4 contains three filler programs which will be used to set up the workspace for the three programs PROG1, PROG2, and PROG3. Included in this chapter are three sample programs to be entered in these program files, replacing the filler programs. This disk also has four empty files for storing data files DATA1, DATA2, DATA3 and DATA4. In addition, it has the rename capability discussed in Chapter 3 and a program that will erase data files.

A data file is composed of units called records. A record contains one or more individual data items. As an analogy, one might consider a filing cabinet as a data file and a folder in the cabinet as a record which is filled with data items. A typical data item would be a number or string of characters such as a name.

A data item does not move directly from its memory (workspace) location to disk, but is moved first to an intermediate location called a buffer. A buffer is a block of memory (workspace) that has been set aside to handle the shuffling of data between the workspace and the disk. A programmer should think of moving data to and from the disk as a two-stage process: In order to move data to the disk, it must first be moved to the buffer and then to the disk. In order to move data from the disk to the workspace, it must first be moved to the buffer and then to the workspace. In OS-65D there are two buffers available, referred to as buffer#6 and buffer#7 (also referred to occasionally as device#6 and device#7). This chapter provides an introduction to the use of these buffers. Additional details will be provided in Chapter Five, Section J, and in Chapter Six.

A. Sequential and Random Files

There are two types of data files: sequential access files and random access files. A sequential data file is a file in which information is stored sequentially, one item right after another, from the beginning to the end of the file. Sequential files would be used to store a large numeric array, or to store information that can be searched sequentially such as names and phone numbers. The program you will enter as PROG1 will access the file DATA1 sequentially.

In many applications sequential files become impractical. For instance, in an inventory application, one would like to be able to quickly access an inventory item for reference or change. This requires the use of a random data file. Random data files differ from sequential files in that groups of entries are combined into records. These records can be randomly (non-sequentially) accessed. For instance, a random data file could have a hundred records. A program could quickly access any one of these records by record number. For example, the contents

of record 58 could be accessed and then the contents of record 72 could be accessed without looking at any of the 14 records in between.

As we shall see later, buffer#6 must be used to implement random files. Either buffer#6 or buffer#7 (or both) can be used with sequential files.

Because random files permit more sophisticated access methods than do sequential files, random files have a more complicated structure. Each record in a random file has a block of the computer's memory reserved for it. Unless intentionally changed, the length of this block is 128 bytes. The record will take up this much room in the buffer and on the disk regardless of how much of the record has actually been used to store meaningful (to the user) information. When the records of a random file have this characteristic, they are called fixed length records. OS-65D does not provide special features that permit implementation of random files with variable length records.

A floppy disk is divided into concentric circles called tracks on which the information is stored. A 5" disk has 40 tracks and an 8" disk has 77 tracks. A track on an 8" disk has 50% more storage capacity than a track on a 5" disk. When a random file is stored on disk the number of records stored on each track of the disk is fixed in accordance with the record length. For 128 byte records, 16 records will fit on each track of a 5" disk and 24 records will fit on each track of an 8" disk. Additional details on diskette formats will be covered in Chapter Five, Section C, and in Chapter Six.

Procedures for changing the record block length and therefore the number of records per track will be discussed later in this chapter. You must observe the record block length carefully. If you write more characters into a record than its length allows, you will be writing over, and hence destroying, material in the next record.

B. The PRINT# and INPUT# Statements

The PRINT# and INPUT# statements move data between the workspace and the buffers. They function in the same manner as the usual PRINT and INPUT commands in BASIC with the exception that they must refer to a specific buffer (either 6 or 7).

The INPUT# statement reads data from a disk file into the workspace. It has the general form:

```
INPUT#buffer number,string and/or numeric variable list
```

For example,

```
INPUT#6,B$,K
```

Elements in the variable list are separated (delineated) by commas. This INPUT acts on data from a file as if the data were

typed in at the keyboard.

In the above example, B$ is a string variable and K is a numeric variable. The INPUT statement treats these two types of variables somewhat differently. When reading numeric values, embedded spaces are ignored. When a non-space character is found, it is assumed to be part of a number. The number terminates on a colon, comma, or carriage return.

When scanning for string items, leading blanks are ignored. When a character which is not a leading blank is found, it is assumed to be part of a string item. If this first character is a double quote (") the item is taken to be a quoted string, and all characters between the first and second double quotes are returned literally as the characters in the string value. This means that a quoted string in a file may contain any characters except double quotes. If the first character of a string is not a double quote, then it is assumed to be an unquoted string constant. The string returned will terminate on a comma, carriage return, or colon.

The PRINT# statement writes data to a file. It has the general form:

```
PRINT# buffer number,expression list
```

The expression list can consist of string variables, numeric variables, and quoted strings, separated by semicolons. For example,

```
PRINT#6,A$;",";N
```

A$ is a string variable, "," is a quoted string and N is a numeric variable, and each item is separated by a semicolon.

Note the "," element in the example. Because of the way the INPUT statement scans the record, specific delimiters (a comma or carriage return as described earlier) must be placed between data items. When leading blanks are not significant and there are no colons or commas in the strings to be written to the file, it is sufficient with the PRINT statement to insert commas between the strings. For example:

```
PRINT#6,X$;",";Y$;",";Z$
```

When leading blanks are significant or there are commas and/or colons in the strings to be written to the file, the output strings need to be surrounded by double quotes. The CHR$ function can be used to generate the double quote character from its ASCII value as in this example:

```
Q$=CHR$(34)
PRINT#6,Q$;X$;Q$;",";Q$;Y$;Q$
```

If a record is written as a long list, commas need to be inserted between each item. Sometimes it is simpler to write each item with a separate PRINT statement. A FOR-NEXT loop is used to

do this in the following example:

```
FOR I=1 TO 1Ø:PRINT#6,A$(I):NEXT
```

Execution of the PRINT statement for each item causes the data to be followed by a carriage return character. There is another advantage to writing each item with a separate PRINT statement. The BASIC input buffer is 71 characters long. Consequently, longer lists are truncated on input.

C. The OPEN and CLOSE Statements

For the most part, the transfer of data between buffers and diskettes takes place automatically. Because there are just two buffers and potentially many data files, the programmer must make an association between a particular buffer and a particular data file. This association is made by the OPEN statement and dissolved by the CLOSE statement.

Before a program can read or write data to a disk file it must first OPEN the file. The general form of the OPEN statement is

```
DISK OPEN, buffer number,"file name"
```

For example:

```
DISK OPEN,6,"DATA1"
```

This OPEN statement associates the buffer we refer to as #6, with the file DATA1, reads the first track of the file into buffer#6, and sets the memory pointers (the counters which point to the current record) to the start of this buffer (to the first record of the file).

A file that has been OPENed needs to be CLOSEd. The general form of the CLOSE statement is

```
DISK CLOSE, buffer number
```

For example:

```
DISK CLOSE,6
```

The CLOSE statement finishes the connection between the buffer and the file name given in the OPEN for that file. It allows the buffer to be used again in another OPEN statement with another data file. To read from a file that has just been written, CLOSE and re-OPEN the file. This sets the data pointer back to the beginning of the file. A CLOSE for a sequential file writes the data that is still resident in the buffer to the disk.

D. A Sequential File Example (Using Tutorial Disk 4)

Boot-up Tutorial disk four and select option 1 to load PROG1. PROG1 (as well as PROG2 and PROG3) has been created with a

buffer area. Several methods for verifying this will be discussed in later chapters. Next, type NEW <RETURN> to clear the filler program. These first two steps will put a single buffer (#6) in front of your workspace. Now that you have your buffer and a clear workspace enter the following program and store it as PROG1. The procedure should be as follows (Serial system users should omit lines 20 and 40; see the Note, end of Chapter 7):

```
OK
NEW

OK
10  REM SEQUENTIAL FILE DEMONSTRATION
20  PRINT!(28):REM CLEAR SCREEN
30  INPUT"Enter data or List data? (E/L)";C$
40  PRINT!(28):REM CLEAR SCREEN
50  IF C$="E" THEN 100
60  IF C$="L" THEN 200
100 REM ROUTINE FOR ENTERING DATA
105 DISK OPEN,6,"DATA1"
110 FOR I=1 TO 3
120 INPUT"Enter a name";N$
130 INPUT"Enter the telephone number";T$
140 PRINT#6,N$:PRINT#6,T$
150 NEXT I
160 DISK CLOSE,6
170 PRINT"DATA STORED"
180 GOTO 30
200 REM ROUTINE FOR LISTING DATA
210 DISK OPEN,6,"DATA1"
220 FOR I=1 TO 3
230 INPUT#6,N$,T$
240 PRINT N$,T$
250 NEXT I
260 DISK CLOSE,6
270 GOTO 30

DISK!"PUT PROG1"

OK
```

After storing the program in PROG1, you should run the program and correct any errors that are found and save the final copy in PROG1. Note that the first thing the program does is to clear the screen. The screen clear command, PRINT!(28), on lines 20 and 40 is one of the new OS-65D V3.3 features that are documented in Chapter 7. It does not function on serial systems.

After the first screen clear the program prompts you for a choice between entering or listing. The first time through, you should enter "E". The program will then ask for a name. After entering a person's name, the program will ask you for a telephone number. Next, you will again be asked to enter a name and so on for three pairs of names and numbers.

Note that the numbers are being stored as string variables. This will allow you to include a space or dash between the first three and last four digits of the telephone number. Note also

that line 140 contains two separate print statements. This has the effect of placing a carriage return character between the name and number entries. This will be important in Chapter 7 when an extension of this example program is discussed. For the present, however, line 140 could be replaced by:

```
140 PRINT#6,N$;T$
```

This would work equally as well.

When the three pairs of names and numbers have been entered, the message

```
DATA STORED
```

will be displayed and you will be returned to the enter or list options. At this point six string variables have been stored in the data file DATA1 (through the use of buffer#6).

Now enter "L". The computer will load the contents of DATA1 into buffer#6, pick up pairs of string variables with the INPUT statement in line 230, and display these pairs on the screen. When this is completed, the enter or list message will again be displayed.

To exit the program, simply press the RETURN key in response to the enter or list message instead of entering "E" or "L". The OK prompt will appear. Now run the menu by entering the command:

```
RUN"MENU"
```

Since you are now back to the menu, you could now use option 5 to clear data file DATA1.

In Chapter 7 a sample program is presented which can be written by simply adding to the sequential file demonstration program just discussed. Therefore, you may want to save this program in PROG1 for future reference.

E. The GET and PUT Statements (for random files)

The advantage of a random file is that any record may be read or written at any time. This is accomplished by executing the DISK GET command before a PRINT or INPUT statement. As will be explained later, the DISK PUT command may be used after each PRINT statement but this is not necessary with OS-65D V3.3.

The GET and PUT commands have the form:

```
DISK GET, n

DISK PUT
```

where n is a record number.

When a GET operation is performed, the file pointer is set to the record number specified by the DISK GET command. Record

numbers begin with Ø. For standard sized records--128 bytes per record and 16 (24 on 8" inch systems) records per track--no disk transfer operation is involved if the record number is 15 (23 on 8" systems) or smaller. Since the entire first track of a data file is transferred to buffer#6 by the DISK OPEN,6,"filename" command, the data file pointer simply goes to the proper memory location in the buffer. If, however, the record number is 16 (24 on 8" systems) or larger, a new track must be loaded into the buffer and then the pointer can be set.

Once the pointer has been set to the proper record, a PRINT statement or INPUT ststement should follow. A PRINT#6 statement following the GET command will store information in the buffer#6 memory locations corresponding to the selected record. Note that the record number itself does not exist in the buffer in direct retrievable form. If the record number will be needed later, it should be stored in the record along with the other record contents.

The DISK PUT command transfers the buffer contents onto the disk. If a program processing random files is to be run on the earlier OS-65D V3.2 Disk Operating System, then the DISK PUT command must follow each PRINT statement or series of PRINT statements. For OS-65D V3.3, however, this is not necessary. In other words, if new information has been PRINTed into the buffer prior to the execution of a GET command which involves a record on a different disk track from the one currently in the buffer, then the DISK GET command will automatically store the buffer contents onto the disk before bringing in the new track. (This is one of the new features of V3.3.)

When an INPUT#6 statement follows a GET command, the contents of the record specified by the GET command is transferred from buffer#6 to the variables in the INPUT statement.

More than one INPUT or PRINT statement may access the same record. For example, both A$ and B$ are written in record 1 by

```
DISK GET,1

PRINT#6,A$

PRINT#6,B$

DISK PUT
```

F. A Random File Example (using Tutorial Disk 4)

Boot-up Tutorial Disk 4 and select option 2 to load PROG2 in the same way you did in the first example. Then type NEW <RETURN> to clear the filler program. As in the first example, these first two steps are done to acquire a single buffer (#6) in front of your workspace. But this time buffer#6 will be used for random file accessing instead of sequential file accessing (shown in the first example). Now that you have your buffer and a clear workspace enter the following program and

store it as PROG2. The procedure should be as follows (Serial system users should omit lines 20 and 40; see the note at the end of Chapter 7.):

```
OK
NEW

OK
10  REM RANDOM FILE DEMONSTRATION
20  PRINT!(28)
30  INPUT"Enter record or List record? (E/L)";C$
40  PRINT!(28)
50  IF C$="E" THEN 100
60  IF C$="L" THEN 300
100 REM ROUTINE FOR ENTERING DATA
110 DISK OPEN,6,"DATA2"
120 FOR R=1 TO 3
130 PRINT"For record number";R;
140 INPUT"enter item name";I$
150 INPUT"Enter number of items";N$
160 DISK GET,R
170 PRINT#6,R;",";I$;",";N$
180 DISK PUT
190 NEXT R
200 DISK CLOSE,6
210 PRINT"DATA STORED"
220 GOTO 30
300 REM ROUTINE FOR LISTING RECORDS
310 INPUT"Enter number of record to be listed";R
320 DISK OPEN,6,"DATA2"
330 DISK GET,R
340 INPUT#6,RN,I$,N$
350 PRINT RN,I$,N$
360 INPUT"List another record? (Y/N)";A$
370 IF A$="Y" THEN 310
380 DISK CLOSE,6
390 GOTO 30

DISK!"PUT PROG2"

OK
```

After storing the program in PROG2, you can run the program and correct any errors that are found and then save the final copy in PROG2. This program will first clear the screen then prompt you for a choice of entering or listing. The first time through, you should enter "E". The program will then automatically begin with record number 1 and ask you to enter the name of an item. (A program like this could be used for inventory record-keeping.) Next you will be asked to enter how many of the item you have. The program will then store this information (including the record number) in buffer#6 with the line 170 PRINT statement and on the disk with the line 180 DISK PUT statement.

You will be asked to enter item names and the number of items three times after which the DISK CLOSE,6 command will be executed and you again will be given the option of entering or

listing.

After you have stored three records of information, respond to the option message by entering "L". You then can choose record 1, 2 or 3 to be listed. Any other choice will result in an ERR#D error message for line 340 because the program tries to pick up information which is not there. (Note: in Chapter 7 you will be shown how to make profitable use of this type of error detection with the new V3.3 TRAP command.)

After you have listed a record, you will be given the choice of listing another record or going back to the enter or list option. To exit the program, simply press the RETURN key in response to the enter or list message instead of entering "E" or "L". The OK prompt will appear. Then run the menu by entering:

```
RUN"MENU"
```

At this point you could use option 5 in the menu to clear data file DATA2.

After you have become familiar with the program as shown, you may wish to modify it to allow more than three records to be stored. This can be done easily by replacing line 120 with an INPUT statement which asks for a record number to be stored in the variable R. Then line 190 will need to be replaced with statements which give you the choice of returning to line 120 for an additional record or closing the file. If you make this modification, you will be able to store and retrieve records 0 through 31 (0 through 47 on 8" systems) in any order. This limit of 32 (48 on 8" systems) records is imposed because only two tracks on Tutorial Disk 4 have been reserved for data file DATA2. In fact, each data file on Tutorial Disk 4 takes up two tracks.

This limitation can be overcome in two ways. First, the size of the data file can be increased. Methods for doing this will be discussed in Chapter 5. The second way is to decrease the size of each record. Suppose 64 bytes (one byte can store one character) is sufficient to store all of the information, including record number, associated with one record. Then using the standard 128-byte record size is wasteful. If 64-byte records were specified, DATA2 could hold twice as many records.

To change the standard record size, and therefore the number of records per track, you can use the POKEs specified in the table at the top of the next page.

BYTES PER RECORD	RECORDS PER TRACK		VALUE FOR POKE 12Ø76	VALUE FOR POKE 12Ø42	
	5" DISK	8" DISK		5" DISK	8" DISK
256	8	12	8	8	12
128	16	24	7	16	24
64	32	48	6	32	48
32	64	96	5	64	96
16	128	192	4	128	192
8	255	255	3	255	255

The two POKEs must be done only after the statement that opens the file. For example,

```
DISK OPEN,6,"filename"
POKE 12Ø42,32
POKE 12Ø76,6
```

will change the record size to 64 bytes and the records per track to 32 on 5" systems and 48 on 8" systems. Remember that the record size must be carefully observed. If you try to write more characters into a record than its size allows, you will be writing over material in the next record!

As a final illustration of OS-65D V3.3 file handling efficiency, you may remove line 18Ø. As mentioned earlier, the DISK PUT command is not needed in V3.3 (but it is in V3.2). Even when many records are being randomly accessed the DISK GET command alone will keep track of disk transfers and correctly update each track.

In Chapter 7, a sample program will be given which can be written by adding to this random file demonstration program. Therefore, you may wish to save this program in PROG2 for future reference.

G. An Example using Two Data Files

Two data files may be used simultaneously by opening one on buffer #6 and one on buffer #7. Remember that the first buffer (#6) can support both random and sequential file accessing, and the second buffer (#7) can support only sequential file accessing. The INPUT and PRINT statements can now be used with either buffer, interchangeably. More than two data files can be used in a program by simply closing and reopening files assigned to one or both of these buffers.

Boot-up Tutorial Disk 4 and select option 3 to load PROG3 in the same way you did in the previous example. Then type NEW <RETURN> to clear the filler program. These first two steps are done this time to acquire two buffers (#6 and #7) in front of your workspace. Now that you have your buffers and a clear workspace enter the following program and store it as PROG3.

The procedure should be as follows: (Serial system users should omit lines 20 and 240.)

```
OK
NEW

OK
10  REM  DEMONSTRATION OF USE OF TWO DATA FILES
11  REM  ONE SEQUENTIAL, ONE RANDOM
20  PRINT!(28)
30  DISK OPEN,6,"DATA3"
40  DISK OPEN,7,"DATA4"
50  K=1
60  FOR I=1 TO 5:REM  ENTER CATEGORIES
70  INPUT "Enter a category";C$
80  IF C$="NO" THEN 240
90  S=K
100 IF C$="END" THEN 210:REM NO MORE CATEGORIES
110 FOR J=1 TO 6:REM ENTER ITEMS FOR THIS CATEGORY
120 INPUT "Enter item name";IT$(J)
130 IF IT$(J)="END" THEN 190
140 DISK GET,K
150 PRINT#6,IT$(J)
160 DISK PUT
170 K=K+1
180 NEXT J
190 PRINT#7,C$;",";S;",";K-1
200 NEXT I
210 PRINT#7,C$;",";S;",";K-1
220 DISK CLOSE,7
230 DISK OPEN,7,"DATA4"
240 PRINT!(28)
250 FOR I=1 TO 5
260 INPUT#7,A$,S(I),E(I)
270 IF A$="END" THEN 300
280 PRINT I,A$
290 NEXT I
300 INPUT "Enter the number of the category";N
310 IF N>I-1 OR N<1 THEN 380
320 FOR J=S(N) TO E(N)
330 DISK GET,J
340 INPUT#6,B$
350 PRINT B$
360 NEXT J
370 PRINT: PRINT: GOTO 300
380 DISK CLOSE,7
390 DISK CLOSE,6
400 END

DISK!"PUT PROG3"

OK
```

This third example program creates a fundamental type of data structure. There are two data files, one containing CATEGORIES and the other containing ITEMS. The data is created by entering a category followed by items in that category. For example, the data file might be a master grocery list. A sample

category might be "FRUIT" with items "APPLES", "PEARS", "ORANGES", etc. The categories are stored sequentially in DATA4 and the items in the random access file DATA3. The last item in each category and the last category entered must be "END". The first time the program is run the categories and items must be entered. Thereafter, respond to the first appearance of "Enter a category?" by NO. A menu will then be displayed listing each of the categories. When a category is chosen the items in that category are displayed. This occurs because the beginning and ending record numbers of the items in a category are stored along with the name of the category.

The program has been written to accommodate five categories of six items each because of the size of the file DATA3 (on 5" systems only 32 standard size records can be stored in DATA3). The items are stored in the random file one after the other as the first portion of the program is executed. No provision is made for adding or subtracting items from the categories without re-running the entire first portion of the program.

To exit the program either enter a category number outside the range or simply press <RETURN> without entering a number.

As with the previous examples, the data files can be erased by returning to the Tutorial Disk 4 menu and choosing option 5. Note: This example will not be referred to in Chapter 7; hence, it is not necessary to save it for future reference.

Chapter 5

Tutorial Disk Five

A. Introduction

Tutorial Disks 1 through 4 are designed to introduce you to the use of the OS-65D operating system. Tutorial Disk 5 is intended to be used as the "system disk" or working disk with version 3.3 of OS-65D. This chapter describes the use of this disk.

When Tutorial Disks 3 and 4 are booted up, the menu displays a listing of the names of the files provided for storing programs and data files. Each OS-65D diskette maintains a special directory file which stores the names and locations of all the files present on the diskette. On Disk 3, when you select one of the menu options 1 through 8, the computer searches through this directory file for the location of the file requested before it loads the program.

On Disks 3 and 4 you were not concerned with where programs were actually stored on the diskette. There were files present which you could use as needed to store your programs. Although this approach simplifies your initial use of the OS-65D system, it does not make efficient use of the storage capacity of the floppy diskettes. Tutorial Disk 5 provides several features which allow you to make more efficient use of the space available on your diskettes. Before we proceed with a discussion of these features, we need to briefly consider the concept of a workspace and the organization of floppy diskettes.

B. The Workspace

When you use the OS-65D Version 3.3 disk operating system with your Ohio Scientific computer, approximately 14K (1 K = 1Ø24 bytes or characters) of memory are occupied by the operating system. The remainder of your computer's memory is available as a workspace. This workspace is the region available to you when you load and run programs.

The most common way to enter a program into the workspace is to enter it through the keyboard. Recall that this was the procedure you used in Chapters 3 and 4 when you entered and tested several sample programs.

It is important to realize that any program stored in the workspace is lost each time the computer is turned off or the break key (or reset button) is depressed. Since some programs involve hundreds of lines of code it is obviously desirable to have some method of permanently storing programs. There are two

common forms of external storage used with microcomputers: cassettes and floppy diskettes. Although cassette based systems are relatively inexpensive, storing and retrieving programs from cassettes tends to be slow and inconvenient. OS-65D is a diskette based operating system which allows fast and convenient storage and retrieval of programs. In the following section we will briefly describe the organization of a floppy diskette.

C. Floppy Diskette Formats

Floppy diskettes are divided into concentric circles called tracks. Each track can be further divided into smaller parts called sectors.

Diskette Size	Tracks per diskette	Track Range
8" diskette	77	Ø - 76
minifloppy	4Ø	Ø - 39

Storage capacity is often described in terms of pages with each page consisting of 256 bytes or characters. Each track of an 8" diskette has a capacity of 12 pages. On the smaller minifloppy each track has a capacity of 8 pages. Thus, using the preceding table, we see that an 8" diskette can store over 2ØØ,ØØØ characters (approximately 3ØØØ characters per track) and a minifloppy diskette can store over 8Ø,ØØØ characters (approximately 2ØØØ characters per track).

Not all of the diskette is available to store user programs. As we shall see in Appendix 1, part of the diskette is occupied by the operating system and utility programs. In particular, track Ø is used to store information which must be present when the system is booted up.

Under the OS-65D disk operating system, all program files are stored on consecutive tracks and occupy a whole number of tracks (e.g., you cannot have a program file which occupies 1-1/2 tracks).

The OS-65D operating system provides two commands for storing and retrieving programs from diskette: the DISK!"PUT and the DISK!"LOAD commands. You have already used the DISK!"PUT command on Disks 3 and 4 to store programs in named files. A complete description of these two commands is given in section H of this chapter. It is possible to load and store programs by track number. The command DISK!"PUT 27" will, for example, store the program currently in the workspace beginning at track 27 and continuing for as many tracks as are necessary to hold the complete program. The major disadvantage of using track numbers is that it requires the user to remember exactly what is on each track.

D. The Menu

Boot up your system using Tutorial Disk 5. The following

menu will be displayed on the screen.

```
OS-65D Tutorial disk five

1 > Directory
2 > Create a new file
3 > Change a file name
4 > Delete file name from diskette
5 > Create blank data diskette
6 > Create data diskette with files
7 > Create buffer space for data files
8 > Single or dual disk drive copier
9 > Enter OS-65D system

Type the number of your selection and depress RETURN?
```

The OS-65D operating system maintains a directory which keeps a record of the names and track locations of all files on each diskette. The first option in this menu, which is discussed in the next section, displays a listing of this directory.

The next menu option discussed will be option 8 which allows diskettes to be copied on either single or dual disk drive systems. You will be shown how to use this option to make a copy of Tutorial Disk 5 for everyday use. The original copy of disk 5 provided with version 3.3 of OS-65D is write-protected to protect its contents from accidental alteration. Options 2, 3 and 4 in the menu make it possible to delete files, change the names of existing files or add new files to the disk. As an application of these options you will be shown how to create a streamlined version of disk 5 with room for user written programs.

Option 9 terminates the menu and places the computer in the BASIC mode. This allows the user to enter his own programs. Finally, options 5, 6 and 7 are designed to simplify the use of data files.

E. The Directory

Option 1 provides a listing of all files currently present on the diskette together with their track locations. Boot up and enter 1 (followed by <CR> of course) in response to the menu prompt. You will be prompted by the message:

```
Directory utility
Directory of which drive?

Type A,B,C or D and depress RETURN <A>?
```

The letters A/B/C/D denote different disk drives. If your computer has a single disk drive, it is called drive A. On a dual disk drive system, the top drive is designated as drive A and the bottom drive is designated as drive B. Drives C and D are applicable only for dual-sided dual disk drive systems. The <A> in the last prompt line means drive A will be selected by default

if you press <CR> without specifying A, B, C or D. If you enter B, C or D the appropriate disk drive will be selected for the directory listing. If you have a single disk drive system, you can simply respond to all such questions by press <CR>. If you select a drive that does not exist on your system (e.g., drive B on a single disk drive system), an error message will be printed and the program will terminate. You will then have to type RUN"BEXEC*" to return to the menu. (Note: although it need not concern you at this time, the DOS Kernel prompt will be changed to the incorrectly selected drive; see Chapter 6 for details.)

If you want the output to appear on the console screen, just depress <CR> (note that <No> specifies the default response to be No). Otherwise, enter "Y" for output to the printer.

The following is the listing that will result with the 5" and 8" OS-65D diskette number 5. The ouput is essentially the same except for the track numbers.

5" Disk

```
   -- Directory --

File name  Track range
----------------------

OS65D3       Ø - 13
BEXEC*      14 - 16
COPIER      17 - 18
CHANGE      19 - 2Ø
CREATE      21 - 22
DELETE      23 - 23
DIR         24 - 24
RANLST      25 - 26
RENAME      27 - 27
SECDIR      28 - 28
SEQLST      29 - 3Ø
TRACE       31 - 31
ZERO        32 - 33
ASAMPL      34 - 34
ATNENB      35 - 35
COLORS      36 - 36
MODEM       37 - 38
COMPAR      39 - 39

  46 Entries free out of 64

Depress RETURN to continue?
```

8" Disk

```
   -- Directory --

File name  Track range
----------------------

OS65D3       Ø -  8
BEXEC*       9 - 11
COPIER      13 - 14
CHANGE      15 - 15
CREATE      16 - 17
DELETE      18 - 18
DIR         19 - 19
RANLST      2Ø - 21
RENAME      22 - 22
SECDIR      23 - 23
SEQLST      24 - 25
TRACE       26 - 26
ZERO        27 - 28
ASAMPL      29 - 29
ATNENB      3Ø - 3Ø
COLORS      31 - 31
MODEM       32 - 32
COMPAR      33 - 33

   46 Entries free out of 64

Depress RETURN to continue?
```

There are several named files currently listed in the directory. The file OS65D3 contains the operating system and the BASIC interpreter. The file BEXEC* contains a sophisticated utility program which is automatically loaded into the workspace and run each time the computer is booted up. It is this program which generated the initial menu and this directory listing. The

display of the directory will remain on the screen until you depress <CR>, at which time the BEXEC* program will again display the menu on the screen. In order to enter a new program it is necessary to use option 9 to terminate execution of BEXEC*. Another file in this directory listing is COPIER. This is also part of the operating system and should not be altered in any way. According to this directory listing, the minifloppy has no tracks free and the 8 inch disk has track 12 and tracks 34 through 76 free. These tracks can be used to store any new programs and data files you create. Appendix 1 contains a complete description of all the program files listed in this directory.

Option 1 can be used to display directory listings for OS-65D diskettes other than Tutorial Disk 5 itself. As an example, the following procedure can be used to obtain a directory listing for Tutorial Disk 3.

Select option 1 (the directory option) from the menu on Tutorial Disk 5 (if you just displayed a directory it will be necessary to press <CR> to return to the menu). When the prompt lines:

```
Directory utility

Directory of which drive?

Type A,B,C or D and depress RETURN <A>?
```

are displayed, replace Tutorial Disk 5 by Tutorial Disk 3 and then enter A or simply <CR>. After you specify whether the listing should appear on the printer, the directory listing on the top of the next page will be generated.

WARNING: You will want to continue using Tutorial Disk 5, so replace Tutorial Disk 3 by Tutorial Disk 5 before you depress <CR> to terminate the display of the directory listing.

5" Disk

```
   -- Directory --

File name   Track range
------------------------

OS65D3        Ø - 13
MENU         14 - 15
PROG1        16 - 18
PROG2        19 - 21
PROG3        22 - 24
PROG4        25 - 27
PROG5        28 - 3Ø
PROG6        31 - 33
PROG7        34 - 36
PROG8        37 - 39

 54 Entries free out of 64
```

8" Disk

```
     -- Directory --

File name   Track range
------------------------

OS65D3        Ø -  8
MENU          9 - 1Ø
PROG1        16 - 18
PROG2        19 - 21
PROG3        22 - 24
PROG4        25 - 27
PROG5        28 - 3Ø
PROG6        31 - 33
PROG7        34 - 36
PROG8        37 - 39

 54 Entries free out of 64
```

If you have used the rename capability on disk number 3, the new names you assigned the files will appear in the above listing instead of the names PROG1 - PROG8. Recall that each of the entries PROG1 - PROG8 are files which have been precreated for you to store programs.

If you will look back at the directory listing for Tutorial Disk 5, you will note that there are currently no files present to store user written programs. Before you begin writing programs using Tutorial Disk 5, you need to make a new copy of Disk 5 with room for user written files.

F. Copying Diskettes

When option 8 in the menu of Tutorial Disk 5 is selected, the program named COPIER is loaded and run (this program is listed in the directory for Tutorial Disk 5 printed above). The program COPIER is capable of copying diskettes on systems with single or multiple disk drives. As an exercise in the use of the copy option it is recommended that you copy the contents of Tutorial Disk 5 onto the blank diskette (the one with the brown label) provided with your 65D Tutorial Set. In the following sections this copy of disk 5 will be used to illustrate the other options in the menu.

When option 8 (the copy option) is selected from the menu, two lines will be displayed on the screen:

```
- Diskette copier -

Copy from which drive (A/B/C/D) ? _
```

You should respond with the letter designating the drive in which you will place the diskette you wish to copy FROM. Usually

the disk to be copied is placed in drive A, so normally you will respond by entering A. Obviously, if you have a single drive system, you must respond by entering A. There is no assumed default in this situation. If you press <CR> without entering a letter, the computer will repeat its question. The next line displayed by the program COPIER is

Copy to which drive (A/B/C/D) ? _

You should respond with the letter designating the drive in which you will place the diskette you wish to copy TO. If you have a single drive system, again respond by entering A. If you have a multiple drive system, selecting a different drive for the TO disk (e.g., B) than for the FROM disk will speed up the copy process, but this is not absolutely necessary. You will next be asked:

What is the last track to be copied (Inclusive) <Ø-39> ? _

In this case you want to copy the entire diskette so you should enter the response 39 (76 on 8" systems). If you were to respond 15, then only the first 15 tracks would be copied. The final question before the copy process begins is:

Are you ready to start copying (Y/N) ? _

Before answering this question, you should place the diskettes you wish to copy FROM and TO in the appropriate drives, check your previous responses and then enter Y for yes if everything is correct. If you are using only one disk drive for the copy process, do not insert the diskettes yet. You will be prompted to insert and remove them at the appropriate times. If you have made any errors in any of your earlier responses, you should answer N and the initial prompt will be redisplayed.

Although we are using a blank diskette in this example, it is possible to copy onto a diskette which has been previously used. The diskette is initialized as part of the copy process.

As the copy process progresses the messages "Reading --","Initializing --" will be displayed on the screen. A sequence of track numbers will also be displayed which shows the progression of the copy process through the tracks. When the copy process is completed, the message:

Please, put the tutorial disk in drive A and depress <RETURN>

will appear on the screen. After you replace the tutorial disk and press <CR>, the program BEXEC* from Tutorial Disk 5 will be reloaded and run and the menu will reappear on the screen.

G. Creating, Deleting and Renaming Files

The menu on Disk 5 provides options which allow the user to add file names to or delete file names from the directory. Another option allows the user to change the names assigned to files in the directory. These three options are discussed in this

section. You should use the copy of Tutorial Disk 5 with the brown label which you just made to experiment with these features. Your original copy of Tutorial Disk 5 is write protected to specifically guard against the type of changes you are about to make.

The order in which the create, delete and rename options are discussed in this section is a matter of convenience. As observed earlier, all of the tracks on the minifloppy version of Tutorial Disk 5 are currently in use. So before you can create any new files, you probably need to know how to delete some of the old ones.

(i) Deleting a file

Deleting a file removes a file name from the directory. Option 4 in the menu for Tutorial Disk 5 allows you to delete files from the directory.

We will illustrate the delete option by deleting several files from the directory of Tutorial Disk 5. Before you proceed, make sure that your copy of Tutorial Disk 5 with the brown label is in drive A. All of the files listed in the directory for Tutorial Disk 5 except for the first three, OS65D3, BEXEC*, and COPIER, can be safely deleted without affecting the operation of the menu. The delete option in the menu does not use the program in the file DELETE. There is a delete capability built into the BEXEC* itself. The file DELETE contains an independent utility program which can be used to delete files from the directory. For users familiar with version 3.2 of OS-65D, recall that the BEXEC* in version 3.2 did not itself have a delete capability. Files could only be deleted under 3.2 by the use of the DELETE program. The programs in the files CREATE, DELETE, DIR and RENAME are discussed in detail in Appendix 1. They are holdovers from version 3.2 of OS-65D whose functions have been incorporated into the BEXEC* in version 3.3. These (redundant) programs are on the version 3.3 disk primarily for the convenience of experienced version 3.2 programmers.

The following series of steps will remove the file named CHANGE from the directory. Select the delete option by entering 4 when the menu asks which option you wish to select. The following prompt lines will appear on the screen:

```
Delete utility

Delete a file on which drive?

Type A,B,C or D and depress RETURN <A>?
```

Since our disk is in drive A we can either enter A or simply press <CR> in which case it will assume A as a default. If you wish to delete a file name from the directory of a diskette in a different disk drive, simply enter the appropriate letter in place of A. Next you will be asked:

Type in the name of the file that you want to delete
and depress RETURN (1-6 characters)?

Respond by entering the name CHANGE. If you make a typing error, a message will probably be displayed saying that whatever name you typed was not found and requesting that you press RETURN to continue. If you type the name correctly there will be a brief delay after which the menu will be redisplayed. When the menu is redisplayed, select option 1 to verify that the file name CHANGE is no longer present in the directory.

If you change your mind after selecting the delete option, you can return to the menu without deleting a file by pressing <CR> when asked for the name of the file you wish to delete.

As an exercise in the use of the delete option, delete the remainder of the files in the directory that are listed after the file COPIER. When you are done the directory listing should only include OS65D3, BEXEC* and COPIER. The resulting stripped version of Tutorial Disk 5 is extremely useful for development purposes. All of the options in the menu can still be used (note the delete option still worked long after the file DELETE was gone) and tracks 19-39 (tracks 15-76 on 8" systems) are now available to store user written programs. In the following this stripped version of Tutorial Disk 5 will be referred to as the development diskette.

(ii) Creating a File

Creating a new file places a new file name in the directory and reserves space on the diskette for a program or data to be stored under the new file name. Option 2 in the menu allows you to create new files. We will describe how to add a new file named PROG1 to the directory of the development diskette.

Boot up with the development diskette and select option 2 by entering 2. The screen will clear and the following question will appear:

```
Create utility

Create a file on which drive?

Type A,B,C or D and depress RETURN <A>?
```

Enter the response A or simply press <CR>. In general you should enter the letter corresponding to the drive containing the diskette upon which you wish to create a new file. After a short wait you will be asked to enter a name for the new file:

Type in the name of the file you want to create and
depress RETURN (1-6 characters)?

Enter the name PROG1 (a file name can be up to six characters in length and must begin with a letter). The computer will check to verify that the name entered is not already present in the directory. If you press <CR> without entering a file name, you will be returned to the menu.

Since we have removed all files after COPIER from the directory on our development diskette, we know that tracks 19-39 are open. The following sequence of responses will place our file on tracks 19-23 (15-76 on 8" systems).

Type in the number of tracks in this file
and depress RETURN (1 to 27)? <1>

Response: 5 <CR>

(The directory will now be searched. The first block of 5 tracks available will be used. If no block of 5 consecutive tracks were available then a message would be printed indicating "no room".)

Do you want to initialize
these tracks (Yes or No) <Yes>?

Response: <CR> or YES <CR>

How many pages per track (1 to 8) <8>?

Response: <CR> or 8 <CR>

(Note: the preceding assumes a minifloppy system. On an 8" system the track numbers will be different and the default pages per track will be 12, rather than 8.)

When you have successfully created this file, select option 1 to display an updated directory listing. Your listing should appear as before with the additional entry for the file PROG1 on tracks 19-24. Press <CR> to return to the menu.

If you make an error when using the create option such as duplicating a file name which is already in the directory, an appropriate error message will be displayed on the screen and you will be asked to press <CR> to return to the menu.

The file PROG1 which you have just created is, of course, an empty file. There is currently no program stored in it. In later sections we will discuss the commands used to load and store programs from and to diskette.

The major difference between Disks 3 and 4, and Disk 5 is that with Disk 5 you are free to create files on an as-needed basis--assigning any name you wish and, more important, making them whatever size you wish. Using Disk 3 is a little like buying clothes for all members of a family at a store which only stocks one size. Even though the clothing may fit the "average" person, most people will find the clothes either too small or too big. When you use Disk 3, the files provided for program storage will

usually either be too big (in which case some diskette storage is wasted) or two small (in which case your program will not fit into the space provided).

It is generally a good idea to maintain a scratch file on all development diskettes which is large enough to accommodate any program which you are likely to enter into the computer. This may save you the embarrassment of typing a long program into the computer only to discover you do not have any place to store it.

Generally, the more memory your computer has, the larger the scratch file needs to be. The size of the program which can be entered into the computer is determined by the size of the workspace. The following table indicates the approximate size of the workspace and the maximum number of diskette tracks required for a file for the standard configuration of several Ohio Scentific computers.

Model	Standard Memory Size	Workspace Size	Diskette Type	Maximum # of Tracks
C1PMF	2ØK	6K	Mini	3
C4PMF	24K	12K	Mini	6
C4PDF	48K	34K	8"	12
C8PDF	32K	18K	8"	6

As you gain experience with the use of your computer, you will begin to develop a sense of how large a scratch file you should maintain. A scratch file with a capacity of 1Ø - 15K is large enough for most programs.

The 5-track file PROG1 which we have just created has a capacity of 1ØK for minifloppy systems and 15K for 8" systems. Next we shall use the rename option of the menu to change the name of the file PROG1 to SCRTCH. It would of course have been easier to have just created the file with the name SCRTCH but that approach would have eliminated this chance to use option 3 in the menu.

(iii) Renaming a File

Option 3 in the menu for Disk 5 allows you to change the names of files in the directory. As mentioned above, we will illustrate this capability by changing the name of the file PROG1 we have just created to SCRTCH.

Select option 3 from the menu by entering 3. You will then be asked:

```
Rename utility

Rename a file on which drive?

Type A,B,C or D and depress RETURN <A>?
```

Enter A or just press <CR>. In general you should enter the letter corresponding to the drive containing the disk on which you wish to change the name of a file. Then you will be asked:

Type in the name of the file that you want to rename
and depress RETURN (1-6 characters)?

Answer PROG1. If you change your mind after selecting option 3 you can return to the menu by pressing <CR> without entering a file name. After a brief wait the following message will appear asking you to enter the new file name:

Type in the name that will replace "PROG1 " in the directory and depress RETURN (1-6 characters)?

Enter the new file name SCRTCH. If you enter everything correctly the menu will be redisplayed. If you make a mistake typing in the old file name, an error message may be displayed since it probably will not be able to find the name you entered in the directory. If this happens try again. When you have changed the file name from PROG1 to SCRTCH successfully, select option 1 again to display an updated directory listing and verify your change.

In the next section we will discuss the commands used to transfer programs back and forth between the workspace and the diskette.

H. The PUT, LOAD and RUN commands

We will discuss three commands in this section. They are the PUT, LOAD and RUN commands. IMPORTANT: These commands cannot be used successfully until option 9 (to be discussed fully in the next section) or option 7 (to be discussed in section J) has been selected and the system has responded with the prompt:

```
OK
```

This indicates that the system is in the immediate mode for programming in BASIC. PUT, LOAD, and RUN are not valid responses to the menu; they can be used only in the immediate mode of BASIC.

(i) The PUT Command

The PUT command is used to transfer a program from workspace to diskette. This command can be used in two forms

```
DISK!"PUT <filename>"        or simply      DISK!"PU <filename>
```

and

```
DISK!"PUT <track number>"    or simply      DISK!"PU <track number>
```

Each of these commands is designed to store onto the disk the program currenty stored in the workspace. When the PUT command is used with a file name, the computer first searches through the directory to determine the tracks assigned to the file name and then begins to write the program out to these tracks (checking to make sure that it does not write beyond the end of the file). When the PUT command is used with a track number, the computer begins to write the program out to diskette beginning with the specified track using as many tracks as necessary to hold the program.

As an example, suppose the workspace contains a BASIC program which you wish to store in the file named SCRTCH located on tracks 19-23 which you created on the development diskette. The original copy of Tutorial Disk 5 is shipped write-protected so the DISK!"PUT commands cannot be used with it. As long as the program will fit within the five tracks in the file, the following commands all have the same effect

```
DISK!"PUT SCRTCH"

DISK!"PU SCRTCH

DISK!"PUT 19"

DISK!"PU 19
```

If the program is too long to fit on the five tracks allocated to the file SCRTCH, then the first two of these commands will result in the error message:

ERR#D

The last two commands will successfully store the program (using as many tracks as necessary). Although this might initially seem desirable, you should realize that there could very well be another file named TOOBAD stored on tracks 24-27 containing a program. If the DISK!"PUT 19" command used more than 5 tracks, the program originally stored in the file TOOBAD would be overwritten and therefore lost.

When the PUT, LOAD and RUN commands are used with track numbers rather than file names, the computer does not consult the directory at all. Because of problems of the type outlined above, it is recommended that whenever possible you utilize named files and use your PUT, LOAD and RUN commands with file names. If you create a sufficiently large scratch file, you should never find yourself in the situation of not having a named file large enough to hold the program in the workspace. As a further precaution, you may want to keep a written record of what is stored on each track of your development diskette.

(ii) The LOAD Command

The LOAD command is used to transfer a program file from a diskette into the workspace. This command can be used in two forms

DISK!"LOAD <filename>" or simply DISK!"LO <filename>

and

DISK!"LOAD <track number>" or simply DISK!"LO <track number>

Each of these commands loads a program stored on the diskette into the workspace. When the LOAD command is used with a file name, the computer must search through the directory to determine the track location of the file.

As an example, if a file named PROG2 is located on tracks 16-18 and contains the program you wish to load any of the following commands will load the program into the workspace:

DISK!"LOAD PROG2"

DISK!"LO PROG2

DISK!"LOAD 16"

DISK!"LO 16

Note that when a program is loaded by track number, it is only necessary to specify the starting track of the program.

Recall that when a file is first created by placing its name in the directory it is an empty file. The LOAD command should not be used on an empty file or an error will occur. The next chapter includes a table listing the common input/output errors that can occur with disk files. Attempting to load from an empty file or track will yield an error. Attempting to load a file name which is not present in the directory (this is usually due to a typing error) will result in a #C error.

The following example shows how to use the DISK!"LOAD command to examine a program stored on diskette. Since there are few programs left on the development diskette, reboot with your original copy of Tutorial Disk 5. The directory listing for this disk listed a file named COLORS. You can examine the program COLORS as follows:

1) Select option 9 in the menu

2) When the OK prompt is displayed, enter the command DISK!"LOAD COLORS"

3) Now enter the command LIST

The LOAD command brought the program COLORS into the workspace from the diskette. The LIST command causes the contents of the workspace to be listed.

Now reboot with your development diskette, select option 9, and try the LOAD command with the file SCRTCH which you created in the previous section. Unless you have stored a program in the file, you should experience an error message.

On disks 3 and 4 the LOAD command was automatically generated by the BEXEC*. For example, selecting option number 1 on disk 3 was tantamount to issuing the command DISK!"LOAD PROG1".

(iii) The RUN Command

The RUN command can be used in any of the following forms:

RUN

RUN"<filename>"

RUN"<track number>"

When the RUN command is used without a file name or a track number, it executes whatever program is currently present in the workspace. Thus, one way of executing a program stored on diskette is to issue the command

DISK!"LOAD <filename>" or DISK!"LOAD <track number>"

followed by the command

RUN

The first command will load the desired file into the workspace and the second command will execute it.

The other two forms of the RUN command combine both of the above steps. The command RUN"PROG1" automatically loads the program stored in the file PROG1 into the workspace and then runs it.

I. Entering the BASIC Mode

With Disk 5, the program BEXEC* controls the operation of the computer until the user selects option 9 or 7. When option 9 is selected, the computer enters the BASIC mode. The control of the program BEXEC* is terminated and you must type NEW to clear the workspace. The screen displays the following message:

```
The system is now open for modification

OK
```

You can now enter, list, or run BASIC programs. The commands discussed previously can be used to LOAD and RUN programs from diskette or you can enter a program directly through the keyboard as described in chapters 3 and 4 and store them on diskette using the PUT command.

As a simple example, choose option 9 from the menu and then enter NEW. Try the commands RUN and LIST. You will find that nothing happens because you have cleared the workspace with the command NEW. Now load the program BEXEC* back into the workspace using the command DISK!"LOAD BEXEC*. If you wish you can examine this program using the LIST command. Since the program BEXEC* is now present in the workspace, it can be executed by entering the command RUN. If you try this you will find that the menu is redisplayed. (You could have accomplished the same thing by just entering the command RUN"BEXEC*".)

You should now type in a short sample program. Be sure to enter NEW before doing so. If you fail to enter NEW, BEXEC* will remain in the workspace and the statements of your sample program will simply mix with and/or replace statements of BEXEC*, resulting in a long program that is probably meaningless. A few sample programs can be found in Chapters 3 and 4 of this manual or in your user's manual. Once you have entered a program and are satisfied that it is working correctly, temporarily save it in the scratch file by using the command DISK!"PUT SCRTCH".

Next we would like to create a file to permanently store your program. The scratch file is probably much larger than necessary. The following sequence outlines the necessary steps:

Step 1. Determine how many tracks are needed to store your program. (The procedure is described in Chapter 3

as well as below.)

After you have temporarily stored your program in the file SCRTCH, issue the command EXIT. The computer will print a message telling you how many tracks are required to store the program in the workspace. Make a note of this number and type RE BA to return to the BASIC mode. If you enter the command LIST after the OK prompt is displayed you will observe that your program is still present in the workspace.

Step 2. Create a new file of the required size

Return to the menu either by rebooting or by entering the command RUN"BEXEC*" and select option 2. This process has, of course, loaded BEXEC*, wiping out the copy of your program which was in the workspace (a matter of no concern to you now because you have saved a copy of your program in the file SCRTCH). Using the procedure described in section G of this chapter, create a new file with the appropriate number of tracks and with whatever name you desire.

Step 3. Store your program in the new file

Return to the BASIC mode with option 9 and load your program back into the workspace from the scratch file by entering the command DISK!"LOAD SCRTCH". Your program can now be stored in the new file by entering the command DISK!"PUT <filename>" where <filename> is whatever name you assigned to the new file.

The scratch file now contains an extra copy of the program. The next time a program is stored in the scratch file it will write over this particular program, but the copy we placed in the new file will still be present.

J. Using Data Files

Disk data files provide a convenient means for permanently storing large amounts of data in a form which can be easily accessed by the computer. Disk data files also provide a means of storing the output from one program in a form which can be used as input to another program. Many business applications involve updating data sets on a monthly basis. In these types of applications it is not feasible to expect all of the data to be entered through the keyboard each time the program is run. Similarly, it would not be feasible to include the data in DATA statements within the program.

Although a diskette may contain several data files, the OS-65D operating system only allows two data files to be open (i.e., in use) at any given time. The reason for this is that the OS-65D operating system designates exactly two special buffer areas, known as buffer#6 and buffer#7, for disk input/output and one of these two buffers must be assigned to each open disk data file. Recall that these buffers were first introduced in Chapter 4, but at that time you did not have to worry about how they were implemented.

(Note: buffer#6 and buffer#7 are frequently referred to as device#6 and device#7 in other OSI manuals.)

These buffers play a crucial role in the input/output processes with disk files. Each of these buffers is capable of storing the contents of one complete track of a data file. When a data file is opened, the first track of the data file is copied into the buffer. The INPUT# and PRINT# program statements introduced in Chapter 4 do their actual input and output from and to this buffer area, rather than directly from and to the disk.

The following table shows the memory locations of these buffer areas under version 3.3 of OS-65D.

	Minifloppy system	8" system
Buffer#6	$3A7E-$427D	$3A7E-$467D
Buffer#7	$427E-$4A7D	$467E-$527D

The DISK!"LOAD command described in section H of this chapter loads the contents of the specified file into memory beginning at $3A7E (the beginning of workspace). Thus, in general, part of the program itself will be loaded into the regions designated for buffer#6 and buffer#7. This presents no problems as long as the program does not attempt to open and use any disk data files. If a program does use disk data files and special arrangements have not been made when the program was originally created, it is possible for part of the program to be destroyed when a data file is opened and the first track is read into the appropriate buffer region.

The possible clash between the data file buffer and the program can be avoided by ensuring that none of the program ends up in the buffer areas. If your program will use one disk file you must ensure that none of the program is loaded into the memory region assigned to buffer#6. If your program will use two disk files concurrently, then you must ensure that none of your program is loaded into the memory region assigned to either buffer#6 or buffer#7.

In Chapter 4 you did not have to worry about establishing space for the buffers because you were using pre-created files for which this had been done already. In the absence of pre-created files, option 7 in the menu of Tutorial Disk 5 provides a convenient method for setting workspace parameters to ensure that a program which uses disk data files does not encroach on the buffer areas.

If you plan to write a program which does not use disk data files at all, select option 7 and respond Ø to the question:

```
    Buffer set utility

Type in the number of file buffers you need
and depress RETURN (Ø,1, or 2) <Ø>?
```

The message:

```
No file buffers are resident
Type in your program and save it on your diskette.

OK
```

will appear indicating that the computer is in the BASIC mode waiting for you to enter a program. As you enter your program, it will be stored in memory beginning at $3A7E, the start of workspace. (Note: Another way to set up an empty workspace with no buffers resident is to select option 9 from the menu and then type the command NEW.)

If you plan to write a program which uses a single disk data file, select option 7 and respond 1 to the question:

```
   Buffer set utility

Type in the number of file buffers you need
and depress RETURN (Ø,1, or 2) <Ø>?
```

The message:

```
A single buffer is now resident
Type in your program and save it on your diskette.

OK
```

will appear indicating that the computer is in the BASIC mode waiting for you to enter a program. Since one buffer has been specified, as you enter your program it will be stored in memory beginning at $427E ($467E on 8" systems), leaving the region occupied by buffer#6 open for use by the disk file.

If you plan to write a program which will use two disk data files, select option 7 and respond 2 to the question:

```
   Buffer set utility

Type in the number of file buffers you need
and depress RETURN (Ø,1, or 2) <Ø>?
```

The message:

```
Two buffers are now resident
Type in your program and save it on your diskette.

OK
```

will appear indicating that the computer is in the BASIC mode waiting for you to enter a program. This time since two buffers have been specified, as you enter your program it will be stored in memory beginning at $4A7E ($527E on 8" systems) leaving the region occupied by buffer#6 and buffer#7 open for use by the disk files.

An alternative method for setting aside data file buffer areas is provided by the special utility program BUFFER. This method is more general than option 7 described above because it permits the creation of buffer areas of any size (not just single tracks). Experienced programmers will find this useful for both data file use and other purposes. The BUFFER program is contained on Tutorial Disk Two. Its use is fully documented in Chapter 8. The utility program CHANGE, discuseed in detail in Appendix 1, provides yet another method for setting aside room in the workspace for data buffers.

Once the proper buffer areas have been established the DISK OPEN, DISK CLOSE, DISK GET, and DISK PUT commands can be used to actually process data files. The use of these commands has already been explained in Chapter 4.

K. Storing Data Files

Options 5 and 6 on the Tutorial Disk 5 menu allow you to create a diskette dedicated to storing data files. When either option 5 or 6 is selected, the following sequence of prompt lines is displayed:

```
    Data disk create utility

Be sure the tutorial disk is in drive A

Depress RETURN to continue ?

Remove your tutorial diskette from drive A and
replace it with your blank diskette.

Depress RETURN to continue ?

Your diskette is now ready for data files.

Remove your blank diskette from drive A and
replace it with your tutorial diskette.

Depress RETURN to continue ?
```

Both option 5 and option 6 initialize the data diskette. Option 5 leaves the directory empty except for a necessary entry DIRECT for the directory file. Option 6 creates five empty files named USERØ1 through USERØ5 which are five tracks long and can be

used as data files or program files. If you try Option 6, RUN the directory to verify the existence of these five files.

Both programs and data can be stored on diskettes created by options 5 and 6 but boot-up is not possible because there is no system saved on these diskettes. Therefore, you must boot-up with any diskette containing a system (e.g., Tutorial Disk 5) and then switch diskettes to load and save programs or data.

L. Conclusions

This completes the tutorial introduction to OS-65D. The following chapters provide reference material for all of the standard and optional features of the various versions of OS-65D. Chapter 6 is an overview of the OS-65D "kernel"--the layer of software that interfaces the actual machine operations and the usual user level of operation that has been discussed in the previous five chapters.

Chapter 7 provides a detailed look at the new features that distinguish OS-65D V3.3 from earlier versions. The style of this chapter is very similar to the earlier tutorial chapters. Although there is no special Tutorial Disk keyed to this chapter, the presentation builds upon example programs that were introduced in Chapter 4 and that the user was advised to store on Tutorial Disk 4 for future reference.

Chapter 8 contains a description of the six programs that comprise the OS-65D Extended Utilities Package. These programs are stored on Tutorial Disk Two. Included are special programs that make it possible to:

1. Create, delete and check for the presence of program beffers;
2. Repack programs by removing REM statements and/or blank spaces;
3. Resequence programs in a variety of ways;
4. Sort data files;
5. Copy data files;
6. Disassemble machine language programs.

The overview of OS-65D concludes with Appendix 1 which includes detailed descriptions, including listings, of all of the utility programs contained on a standard OS-65D system diskette.

Chapter 6

Overview of OS-65D

A. Introduction

The operating system is the software that controls all the hardware and software components of the computer. The hardware that is controlled includes:

- the terminal (for serial systems)
- the keyboard and monitor (for video systems)
- the disk drives
- RAM memory allocation
- the printer

The software that is controlled includes:

- the I/O drivers
- the BASIC interpreter
- the Assembler/Editor
- the Extended Monitor
- utilities written in BASIC

Disk storage of programs and data is a major element in using the computer and much of the operating system's functions are devoted to disk I/O. For this reason, the operating system of a computer may be called a "Disk Operating System" or DOS.

The operating system must interface users who prefer to communicate in clear language to hardware which needs explicit commands in numerical and symbolic form. To bridge this gap and still retain flexibility, the operating system was built in a hierarchal form with four layers:

1. I/O drivers and distributor
2. OS Kernel
3. BEXEC*
4. Utilities

The I/O drivers and OS Kernel are written in machine language. The OS Kernel is, however, a self-standing operating system with its own commands directly usable by the computer operator. BEXEC* and the utilities are written in BASIC. BEXEC* is the principal interface between the user and the computer. When the computer is turned on and booted up with an OS-65D disk, the BEXEC* program is in final control, displaying a menu and awaiting the user's commands.

BASIC was the language chosen for these programs for several reasons. It encourages use of clear language instructions and explanations. It has the powerful test and branch instructions needed for control implementation and it is widely understood by end users who may want to modify some features of the operating system to meet local needs.

For most users, the most visible parts of the operating system will be BEXEC* and the DISK!"cmd" commands in BASIC. But the Kernel is the power behind the throne, so this chapter is organized around the features present in the Kernel and how they are used to control various systems in the computer. In many cases we will refer the reader to other OSI manuals for details.

If you are a new user of OSI computers, you should refer to the earlier Tutorial chapters and the appropriate Introductory Manual for a description of start up procedures using a OS-65D disk. The manuals are:

C1P and C1P MF Introductory Manual
C4P MF Introductory Manual
C4P DF Introductory Manual
C8P DF Introductory Manual
Professional Computers Set Up and Operations Manual

The above manuals may describe a menu from BEXEC* that is somewhat different from that displayed by your computer. If so, refer to the supplementary documentation for your version of OS-65D.

B. Memory Allocation

The software used on an OSI computer may be divided into three general categories: System routines (including I/O drivers), Transient Language Processors (BASIC, Assembler, etc.), and Application programs (both user written and utilities for OSI). Each category is stored in a specific place, so a memory map helps clarify the possible cooperation and conflict in and between the various categories.

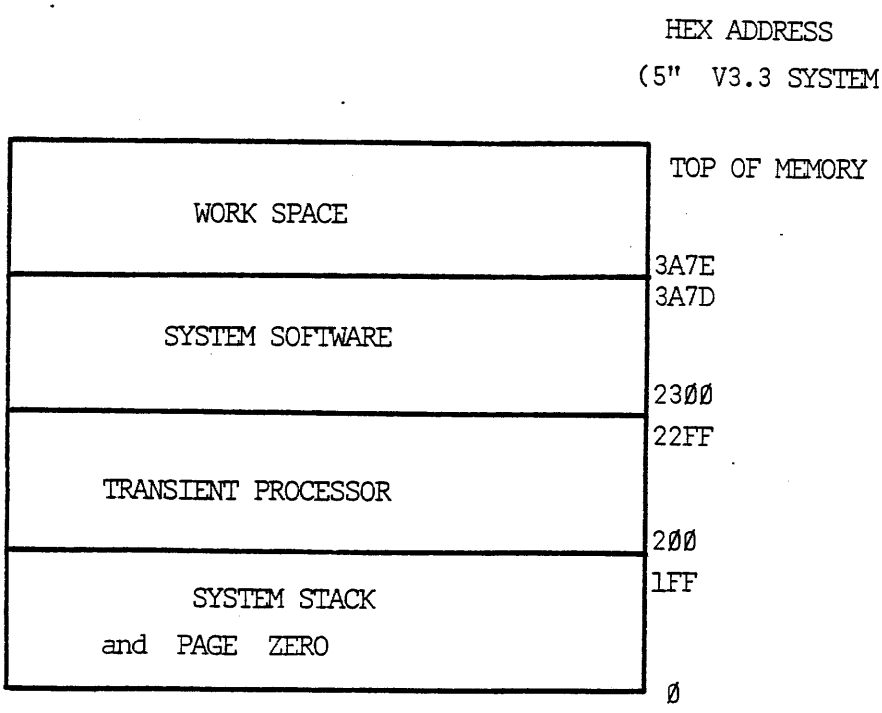


Figure 1: Idealized Memory Map

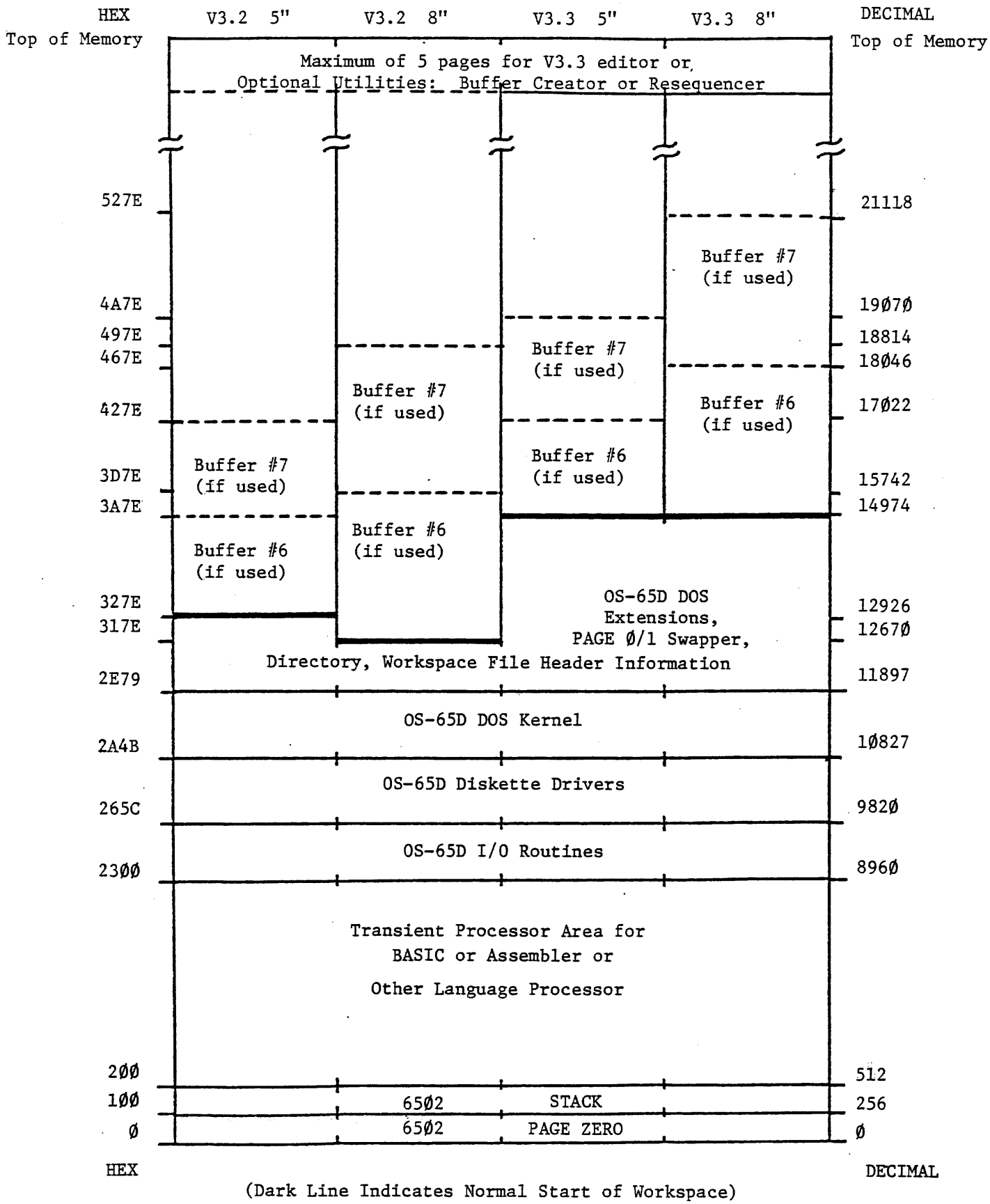
This map is called "Idealized" because various details (such as the addresses at which boundaries occur) will differ depending on the model of computer and version of software used. Figure 2 gives a more detailed memory map for the various cases.

Let us study the Idealized map. Once loaded, the Systems software will remain essentially intact. The other two major areas, Transient Processors and Workspace, are more often changed.

The most common resident in the Transient Processor memory is the BASIC interpreter which must be present if BEXEC* or indeed any program written in BASIC is to be run. Another important resident (replacing BASIC) may be the Assembler/Editor and the Extended Monitor, which fit simultaneously into the Transient Processor memory, with control being given to one or the other. (The Assembler/Editor and Extended Monitor are not furnished with some versions of OS-65D, in which case they may be purchased together from OSI as an optional package).

The workspace may be utilized in several ways. Most commonly, one writes source code in BASIC or assembly language which goes into a source file starting at the lowest address in workspace and expanding upward. In the case of assembler source code one may compile it into machine language code, placed at some higher point in workspace. Refer to the Assembler/Editor and Extended Monitor Reference Manual for

Figure 2: SYSTEM MEMORY MAPS



(Dark Line Indicates Normal Start of Workspace)

details on how to save the machine language code to disk and to read it back in at the start of workspace for execution.

If your BASIC program will use data files, then you need to create one or two data buffers (called buffer #6 and buffer #7) between workspace and the memory holding the operating system, offsetting the start of workspace to a higher address. You can accomplish the creation of these buffers and other changes in the workspace allocations with the help of the CHANGE utility. Likewise, the highest portion of workspace memory may be given to a buffer for indirect files (see the BASIC Reference Manual). Certain transient utilities, if loaded, are located in the highest memory above workspace. These include the V3.3 Editor and the Resequencer and Buffer Creator programs (see Chapter 8).

C. Kernel Commands

The Kernel commands can be divided into three groups:

Control: passes control to various language processors such as BASIC or Assembler.

I/O Distributor: sets flags to enable/disable peripheral equipment. Defines addresses of the "memory" I/O driver.

Disk: Reads and writes to disks.

The next three sections will discuss these major areas of activity for the Kernel and the specific commands used to accomplish its tasks. But first the general format for all Kernel commands must be specified.

General Command Format:

- Commands may be up to 17 characters long.
- Words in commands may be abbreviated to two characters because all other characters beyond two and up to the blank are ignored. For example, "RESTART BASIC" can be written "RE BA".
- File names must start with a character "A" through "Z" and can be no more than six characters long.
- Arguments in the command must be punctuated as shown, must not contain spaces and must have a digit everywhere a letter symbol is shown. The digits are decimal by TT (track number) and S (sector number). The digits are hexadecimal for N,M (in addresses or I/O device masks) and P (number of pages).

The Kernel commands may be called from BASIC using the format DISK!"cmd" and from the Assembler/Editor or Extended Monitor by using the format !cmd. In each case, cmd stands for a Kernel command obeying the format laid out in the previous paragraph. Any command that doesn't obey these syntax rules will fail and an error message will be issued:

ERR #7

This means "SYNTAX ERROR IN COMMAND LINE".

D. Transfer of Control

Figure 3 diagrams the paths for passing control from one software system to another. Notice that the Kernel is a central position in this diagram, the more so since some of the paths that seem to bypass the Kernel, say between the Assembler/Editor and the Extended Monitor, actually are using commands "borrowed" from the Kernel. Any command preceded by a "!" mark (except

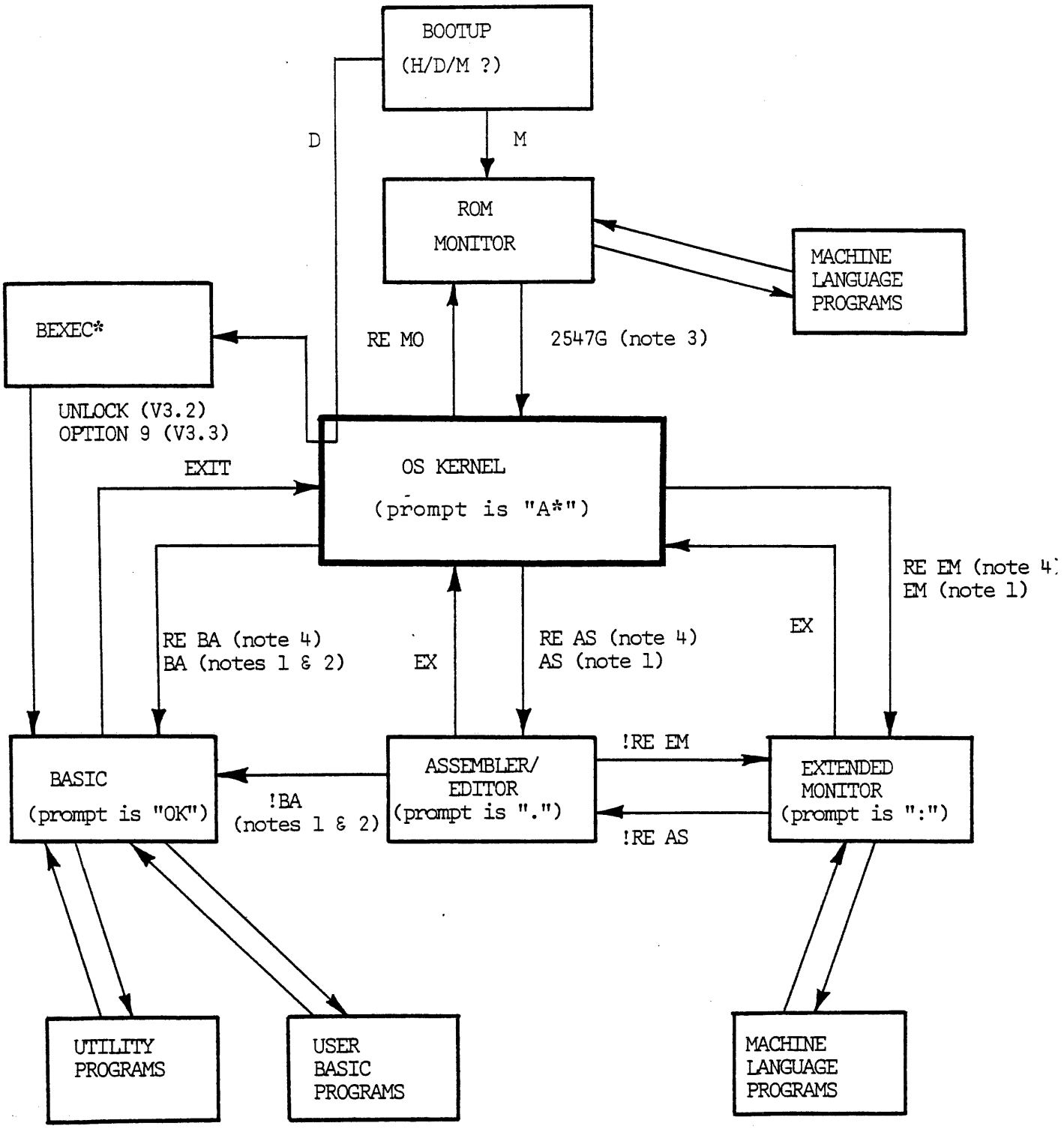


Figure 3 Transfer of Control

those found in a PRINT! command) is actually a Kernel command issued from another program. One returns to the Kernel from any other language by the EXIT command (which can be shortened to EX if you are not in BASIC). One leaves the Kernel and re-starts a language processor (if it is resident in memory) by the command RE XX where XX is BA, AS, EM or MO for BASIC, Assembler/ Editor, Extended Monitor or Monitor in ROM, respectively. "RE" stands for "RESTART". Note that the Kernel is always resident when the operating system has been loaded, but the transient language processors and/or BEXEC* may need to be loaded from disk. The following notes pertain to Figure 3. They should be carefully read as they concern initialization of BASIC and recovery after pushing the BREAK key.

Note 1: These commands will load the specified language processor from disk. The resident language processor will be overwritten.

Note 2: After BASIC is read from disk through the OS Kernel command BA (or !BA) the BASIC commands NEW and LIST are disabled. Execute BEXEC* to unlock these commands. (Remember, loading and executing BEXEC* will destroy any other source code you have in the work area. Put any useful source programs on disk before loading BEXEC*.

Note 3: You can enter the ROM monitor from the Kernel and then return to the Kernel by loading the address $2547 in the monitor address display and depressing G (for "GO"). But you cannot warm start the computer after depressing the BREAK key by the path M to the monitor then $2547G to the Kernel because the act of BREAK leaves some memory locations needed by the operating system with indeterminent contents. The only possible entry to the operating system after BREAK is the cold start made by choosing "D" from the "H/D/M?" menu. This initializes the system and loads and runs BEXEC* which means the contents of the workspace present at the time the BREAK key was depressed are overwritten.

Note 4: These commands can be used only if the specified language processor is resident. For example, if you EXIT from BASIC, the command RE AS will not work; AS must be used.

E. Input/Output Distributor

The computer's major I/O devices are enabled/disabled by a distribution system. This system is described in your system manual (e.g., Appendix Q of the C4P or C8P Users Manual). The devices are:

Flag Setting			BASIC Device #	Device	
Bit	Hex	POKE		Input	Output
-	$ØØ	Ø	9	Null	Null
Ø	Ø1	1	1	Serial Port	Serial Port
1	Ø2	2	2	Keyboard on 54Ø Board	Video on 54Ø Board
2	Ø4	4	3	UART on 43Ø Board	UART on 43Ø Board
3	Ø8	8	4	Null	Parallel Port on CA-9 Board
4	1Ø	16	5	Memory	Memory
5	2Ø	32	6	Disk Buffer #6	Disk Buffer #6
6	4Ø	64	7	Disk Buffer #7	Disk Buffer #7
7	8Ø	128	8	Serial Ports on CA-1ØX Board	Serial Ports on CA-1ØX Board

There are two ways of referring to a given device by numbers. BASIC refers to them by the BASIC device numbers 1 to 9 in such commands as PRINT #6, etc.

The machine keeps track of the enable status of each device by flags:

I/O Distributor Input Flag $2321 (8993 decimal)
Output Flag $2322 (8994)

Each device is assigned one bit in the flag as listed in the table above. The first three columns contain the same flag information in three different formats: the bit position number in the Distribution Flag, the resulting value of the flag byte expressed in hex notation and finally, the same expressed in decimal notation (suitable for POKEing from BASIC). More than one bit may be set in a flag at the same time. For example, one could enable output to the serial port

and to disk buffer #6 simultaneously by setting bits Ø and 5 in the output flag. It then would appear as:

```
ØØ1Ø ØØ1 in binary
     $21 in hex and
      33 in decimal.
```

Notice the value 33 is obtained by adding the individual flag values:

```
$Ø1 + $2Ø = $21 hex or
1 + 32 = 33     decimal.
```

This flag setting could be accomplished by:

```
IO ,21          (in the DOS mode)
```

or

```
POKE 8994,33
DISK!"IO ,21"   (in the BASIC mode)
```

Description of I/O Devices

We identify each device by its BASIC reference number:

9. The Null Device: The null device permits writing programs without having to worry about the characteristics of physical devices. You may read or write to the null device with no effect occurring unless the program expects "handshaking" signals, in which case the computer may hang. For example: INPUT #9,A$ will cause a hang.

1. The Serial Port: This port is configured in various ways on the different models of computers. It may drive a serial printer, a terminal, a tape cassette, or modem.

2. The Keyboard and Video on the 54Ø Board: The standard input and output of video systems.

3. The UART on the 43Ø Board: This is a serial port on some older systems.

4. The Parallel Port: This is a centronics type parallel port on the CA-9 board.

5. The Memory: Memory can also be treated as an I/O device. There are two pointers which specify the current location in memory for input or output respectively as device 5:

Input pointer	$238A (9Ø98 decimal)	lo byte
	238B (9Ø99 decimal)	hi byte
Output pointer	$2391 (91Ø5 decimal)	lo byte
	2392 (91Ø6 decimal)	hi byte

To use device 5 as an input, set the contents of the input pointer to the first address desired. Then after each input of a byte, the pointer will be automatically incremented. For output, the output pointer should be treated similarly: initialized and then automatic incrementation will occur as data is read out. The pointers may be conveniently initialized by using the Kernel command:

MEM NNNN,MMMM NNNN = input pointer in hex
MMMM = output pointer in hex.

6. The Data Buffer #6: This buffer can be created by using One of several utility programs. See Chapters 4, 5 and 8 for details.

7. The Data Buffer #7: See above.

8. The CA-1ØX Serial Ports: See literature supplied with this device.

F. Disk Usage

(i) Tracks

A brand new blank disk has no information in magnetic form on it. The first step in using such a disk is to format it by executing the INIT command. The result is a disk with "tracks": rings of magnetic code. The tracks are numbered from Ø to 39 on 5" disks and Ø to 76 on 8" disks. Each track begins with a track header containing information about the track.

Track number zero has a unique header which is read by a routine in ROM called by choosing "D" from the menu "H/D/M?". This track header starts one millisecond after

detection of the index hole in the disk. The header has three bytes:

```
Load address hi byte
Load address lo byte
number of pages on track zero.
```

The remaining tracks also have the one millisecond delay and then have a 4 byte header:

```
$43
$57
track number as two digits in BCD (Binary Coded Decimal)
$58
```

ii. Sectors

For bookkeeping purposes, tracks are divided into one or more sectors. Each sector has a 3 byte header and a 2 byte trailer and in between has an integer number of pages of data (1 page = 256 bytes):

```
header:   $76
          sector number in binary
          sector length (pages) in binary

trailer:  $47
          $53
```

On a 5" disk, the maximum number of pages per track is 8, so you could have one sector with eight pages of data in it (plus 5 bytes for header and trailer). If you have more than one sector on a track, you can store at most seven pages of data, because the extra bytes required for sector headers and trailers "eat into" the last page of data. On an 8" disk, you can have one sector with 12 pages, or several sectors whose page lengths add up to 11 or less pages.

iii. Fundamental Disk Commands

The Kernel contains nine commands which implement disk I/O using the track and sector formats just described. (There are other commands for disk I/O of source and data files using named and unnamed files. These will be described in part (iv) and (v) of this section. The fundamental commands are:

SELECT X	Select disk drive X where X=A, B, C or D
HOME	Move the current drive's head to track Ø and reset track count to zero
INIT	Initialize whole disk. Erases all information on the disk
INIT TT	Initializes track TT. (Each T is a digit Ø to 9.)
DIR TT	Prints sector directory for track TT. Gives each sector number and its length in pages
D9	Used on 8" systems to disable error 9. Required to read some files from V1.5 and V2.Ø of OS-65D
SAVE TT,S=NNNN/P	Reads P pages starting at NNNN in memory and writes them to sector S of track TT

Note: This command finds track TT and on it the sector with number S-1. Then it writes P pages on the disk, calling it sector S. If there was already a sector S written on track TT, it will be overwritten. If this previous sector S was shorter than P pages, then SAVE will continue writing on into sector S+1. No error message will be issued to warn you of this.

CALL NNNN=TT,S	Load contents of track TT, sector S into memory starting at NNNN. Does not write track or sector headers or trailers into memory
EXAM NNNN=TT	Examines track TT by loading the entire contents, including headers and trailers into memory starting at NNNN

EXAM is a diagnostic tool, allowing examination of headers, etc., as well as the contents of a track, so NNNN is often chosen as video memory. Warning: Occasionally some extra characters will appear before the track and/or sector headers. These characters are often $FF and are not really on the disk. They are due to disk I/O hardware being not yet in sync as the read starts. This is a property of the EXAM command that is not shared by the other read/write commands, and results from a different use of timing loops by the EXAM command. The headers and the data in the file are properly read.

iv. Source Files

When you write a BASIC or assembly language program, the resulting lines of text are called "source code". For source code in assembly language, the assembler will translate the source program into "object code" which is in a form to directly instruct the microprocessor. Although source code can be saved to a disk by using the commands presented in the previous section, these files are so common that some special commands are provided in the Kernel for loading and saving them.

Source code is stored starting at the beginning of workspace, immediately following a 5 byte header:

```
source file header:  start address (lo byte)
                     start address (hi byte)
                     end address   (lo byte)
                     end address   (hi byte)
                     number of tracks required
```

For example, if the source file used neither buffer #6 nor buffer #7 and the system was a 5" V3.2 system, then the source file header would be in locations $3279 through $327D, then the actual source file would start in $327E. (see figure 2)

The format for saving source files on disk is a special case of the general "sectors" format given in the previous section. No choice is allowed in the number of sectors or their length. Each track has one sector that is eight pages long (12 pages for 8" disks). Some number of whole tracks is used to save each source code file. The simplest case saves by track number using these commands:

PUT TT	puts source file on disk, starting at track TT
LOAD TT	loads source file from TT into workspace
XQT TT	loads source file from TT into workspace and transfers control to its first byte, treated as machine language object code

In each case, the smallest number of whole tracks that will hold the source file will be used. This information is obtained from byte 5 of the source code header and printed for you if you enter the Kernel from BASIC (using the EXIT command) or from the Assembler/Editor (using the EX command). You must be careful to choose TT at a spot where sufficient unused tracks exist on the disk, as PU TT will overwrite anything that is there, giving no error messages if named or unnamed files are present. (A major reason for doing file handling from such utilities as CREATE is to provide protection of named files through checking the DIRECTORY.)

The Kernel can use file names when referring to source code files:

PUT FILNAM	puts source code file from workspace onto disk under the name FILNAM
LOAD FILNAM	loads file FILNAM to workspace
XQT FILNAM	loads FILNAM as object code to workspace and passes control to it

Again, an integer number of whole tracks on the disk are loaded by PU FILNAM to workspace. The use of a file name to designate the tracks to be loaded to memory or put on disk is a convenience and provides some safeguards. Two error messages involve named files specifically:

ERROR #C: CAN'T FIND THAT NAME IN DIRECTORY

ERROR #D: READ/WRITE ATTEMPTED PAST END OF NAMED FILE!

Foresight is required, however. The file must be CREATEd before it is used the first time. It is recommended that a large file (perhaps called SCRACH) be kept on a disk to receive source files when you have forgotten to create a file beforehand.

Files can be created using one of the CREATE utilities provided in OS-65D. The utility puts the file name in a DIRECTORY kept on track 12 (track 8 for 8" disk), and reserves a specified number of tracks for it.

Sectors 1 and 2 on track 12 hold the directory. Each entry requires 8 bytes in this format:

file name (six characters)
first track of file (in BCD)
last track of file (in BCD)

The DIRECTORY can hold more entries (64 for a 5" disk) than the disk can hold files (38 maximum for a 5" disk, at one track per file and excluding tracks Ø and 12).

The CREATE utility in V3.2 initializes the tracks (thereby erasing them) but the create choice (no. 2) in the V3.3 menu allows you to leave the tracks as-is, so you can name a file you earlier saved by track number using the PU TT command.

v. Data File Handling

There is one further type of file read and write supported by OS-65D. It handles data files. It is not a part of the Kernel, but rather is in BASIC. We mention it here for completeness. The two commands

```
DISK OPEN,BN,"FILNAM"          BN=6 or 7
DISK CLOSE, BN
```

are used with data buffers #6 and #7. These buffers must be created beforehand by carving some territory out of workspace using one of the CHANGE utilities. These commands, along with PRINT #BN,exp and

```
INPUT #BN,exp
```

service sequential files. Two more commands

```
DISK GET,RN and     (RN=record number)
DISK PUT
```

allow use of random files.

Data files are outlined in the BASIC Reference Manual and extensively discussed, with examples, in Chapters 4 and 5 of this manual.

G. Kernel Command Summary

To enter Kernel:

	Command
From BASIC	EXIT
From Assembler/Editor	EX
From Extended Monitor	EX
From Monitor in ROM	2547G

To leave Kernel:

	Restart	Disk Read and Start*
To BASIC	RE BA	BA
To Assembler/Editor	RE AS	AS
To Extended Monitor	RE EM	EM
To Monitor in ROM	RE M0	--
To Machine Language	GO NNNN	

(*Must be used carefully; see notes 1 and 2 on page 53)

Input/Output Distributor Commands

IO NN,MM	NN = mask for Input device flag MM = mask for Output device flag
IO NN	
IO ,MM	
MEM NNNN,MMMM	Memory (device #5, flag ØØØ1ØØØØ) NNNN = input pointer, MMMM = output pointer

Disk Commands

SELECT X	Select disk drives A, B, C, D
HOME	Moves current drive's head to track Ø
INIT	Initializes whole disk
INIT TT	Initializes track TT
DIR TT	Sectors and length (pages) of track TT
D9	Disable error #9 (8" disks)
SAVE TT,S=NNNN/P	Save to track TT, sector S, P pages from NNNN
CALL NNNN=TT,S	Load to NNNN, sector S of track TT
EXAM NNNN=TT	Dump to NNNN all of track TT including headers
LO TT	Load file starting at track TT to workspace
PU TT	Put file in workspace on disk starting at track TT
XQ TT	Load object file starting at track TT to workspace, execute
LO FILNAM	Load file FILNAM to workspace
PU FILNAM	Put file FILNAM on disk
XQ FILNAM	Load object code FILNAM to workspace, execute

Error Messages from the Kernel

1 - CAN'T READ SECTOR (PARITY ERROR).

2 - CAN'T WRITE SECTOR (REREAD ERROR).

3 - TRACK ZERO IS WRITE PROTECTED AGAINST THAT OPERATION.

4 - DISKETTE IS WRITE PROTECTED.

5 - SEEK ERROR (TRACK HEADER DOESN'T MATCH TRACK).

6 - DRIVE NOT READY.

7 - SYNTAX ERROR IN COMMAND LINE.

8 - BAD TRACK NUMBER.

9 - CAN'T FIND TRACK HEADER WITHIN ONE REV OF DISKETTE.

A - CAN'T FIND SECTOR BEFORE ONE REQUESTED.

B - BAD SECTOR LENGTH VALUE.

C - CAN'T FIND THAT NAME IN DIRECTORY.

D - READ/WRITE ATTEMPTED PAST END OF NAMED FILE!

H. Utility Programs

A variety of different utility programs are available to the OS-65D user. Appendix I contains detailed descriptions and listings for all of the utility programs that appear on a standard OS-65 diskette. Many of these utilities have been discussed previously in Chapter 5. Recall that Tutorial Disk 5 (the V3.3 system disk) incorporates a variety of utility program functions into its BEXEC* program.

A powerful BASIC program line editor is standard in OS-65D V3.3 Its use is described in the next chapter, which also describes the other special features of V3.3.

Finally, a collection of utilities called the OS-65D Extended Utilities is discussed in Chapter 8. These programs can be used to make changes in program files (create buffers, resequence, remove blank spaces and REM statements). There are also utilities for sorting and copying data files and disassembling machine code programs. The Extended Utilities are stored on Tutorial Disk 2.

Chapter 7

New Features in OS-65D V3.3

This chapter presents the new features of V3.3 that extend V3.2. The previous chapter is an outline of features common to the current versions of OS-65D and should be read first by persons not yet familiar with OS-65D.

Although there are many new features in OS-65D V3.3, it is upward and downward compatible with V3.2 for BASIC programs and data files which do not incorporate the special commands unique to V3.3. Even though the workspace in V3.3 has been moved to $3A7E from $327E in V3.2, the V3.3 operating system automatically makes the necessary adjustments whenever a file is loaded or saved. The only problems that may arise involve machine language programs. If a machine language program is to be transferred, the code will not be loaded to the same addresses in workspace with V3.3 as it was with V3.2. Thus, relocation is required.

A. Cursor and INPUT prompt

When the system is booted up with V3.3, the appearance of a flashing square cursor on the screen announces that you are in V3.3. Earlier versions of OS-65D used a non-flashing underline for the cursor.

Both the form of the cursor and the flashing feature are controllable by POKE commands. The command

```
POKE 13Ø26,nn
```

where nn is the decimal value for one of the standard OSI

graphics characters will change the cursor into that character. In particular, POKEing 32 into 13Ø26 will replace the cursor with a blank and thereby eliminate it. Memory location 13743 controls the cursor flashing. This feature can be turned on or off by using one of the two commands listed below:

POKE 13743,44 Disables flashing

POKE 13743,32 Enables flashing

When an INPUT statement is used in a program the "?" prompt indicates that the computer is waiting for input. If you wish to change this prompt, the command

POKE 2797,nn

can be used. As was the case for the cursor, nn is a decimal number corresponding to one of the graphics characters. The question mark is specified by nn = 63.

B. Keyboard Encoder and Video Display

A new routine has been written for input from the keyboard and output to the video display. Keyboard and video on a memory mapped video system now emulate a Hazeltine 142Ø video terminal. The OSI polled keyboard acts like a normal typewriter when used with V3.3. The SHIFT LOCK key acts as a "caps only" key. That is, the SHIFT LOCK key may be raised and the keyboard used as a normal typewriter keyboard. Depressing REPEAT and any other key causes repetition of the key functions. Depressing RUB OUT erases the last character typed in and moves the cursor one space to the left. Simultaneously depressing RUB OUT and REPEAT erases rapidly to the left.

The operative codes of the I/O driver have been altered to allow better control of the screen features of OSI computers. There are codes to control character size, print window size, cursor position, screen clearing, color formatting, screen information transfer to workspace memory, and output to a printer. These codes can be placed in the PRINT statements of BASIC programs and will be described fully later.

In addition to program control of the computer output, some direct keyboard control of the screen display is now possible. By holding down the ESC key and depressing one of the number keys, the display can be altered. The changes which are produced are listed below:

(ESC) 1 Clears screen; homes cursor to upper left; produces "wide character" display

(32x32 on C4P and C8P machines; 24x24 on C1P)

(ESC) 2 Clear screen; homes cursor; produces "narrow character" display

(32x64 on C4P and C8P machines; 12x48 on C1P)

(ESC) 3 Homes cursor to upper left

(ESC) 4 Clears to end of screen (memory of workspace is not altered)

(ESC) 5 Moves cursor up one line

(ESC) 6 Moves cursor down one line

(ESC) 7 Inserts line (lower lines scroll down)

(ESC) 8 Clears line (memory of workspace is not altered)

(ESC) 9 Turns color off

(ESC) 10 Turns color on*

(NOTE: These commands do not work on serial systems; see the "Note to Serial System Users" at the end of this chapter.)

*The color control is for text sent to the screen by way of the keyboard for PRINT statements. It does not affect graphics characters placed on the screen by way of POKE commands. Furthermore, before the color feature can be turned on for the first time after turning on your computer, the wide character or narrow screen format must be specified. This can be done either by depressing ESC and 1 or 2, or by entering one of the two screen formatting PRINT commands by way of immediate mode or a program. These PRINT commands will be described later.

IMPORTANT: These keyboard commands rearrange the screen display but have no effect on a BASIC program in workspace. In particular, clearing a listed program line with (ESC) 8 does not remove it from the BASIC program! Note that (ESC) 1 and (ESC) 2 are convenient screen clear commands from the keyboard.

At the end of this chapter is an overlay which may be cut out and placed along the top row of keys on the keyboard for your reference.

C. Enhanced BASIC

A number of changes have been made in BASIC to improve its flexibility and ease of use. These include upper and lower case interchangeability, line editing, a TRAP command, numerous new PRINT commands for screen formatting and printer control, and new file handling features.

i) Upper and lower case interchangeability

First, it is important to point out that V3.3 does not distinguish between upper and lower case text. Programs may be entered and edited in lower case. Variable names and all commands may be in lower case. String comparison commands will not distinguish between lower and upper case. For example, the statement

```
IF Y$ = "Y" THEN 12Ø
```

will be evaluated true if Y$ = "y" or "Y".

ii) The BASIC line editor

Version 3.3 contains a line editor which functions both for a line being typed in the immediate mode and for lines of a written program. You can edit the current line of text or you can call up by number any line from a stored BASIC program for editing. The cursor can be moved to any point in the line using the cursor control commands listed at the end of this section. A character can be inserted at the location of the cursor (to the left of the character flashing alternately with the cursor) by simply typing the character. RUB OUT will erase the character under the cursor and close the line from the right. RUB OUT with REPEAT "eats up" the line quickly.

If you decide you do not want to keep the edited form of the program line, you can escape from the altered line with the (SHIFT)P command, leaving the original line in workspace. Depressing RETURN writes the entire edited line into workspace, no matter where the cursor was located.

As stated earlier, the above features are applicable to any line you are typing in the immediate or program writing mode of BASIC. If you wish to recall for editing line number nn of a BASIC program in workspace, enter EDITnn or !nn. The line will be displayed on the screen with the cursor at the right end. After editing the line

and storing it (by depressing RETURN), you can call the same line back for re-editing by simply entering !!. Or you can call the next line in the program by entering ! (without a line number). A complete list of editing commands is given below:

(CTRL)H	Moves cursor one space to the left (non-destructively)
(CTRL)P	Moves cursor one space to the right (non-destructively)
(CTRL)F	Moves cursor to the front of the line
(CTRL)R	Moves cursor to the rear of the line
(CTRL)I	Moves the cursor (non-destructively) forward to the next tab position (i.e., positions 1, 8, 15, 22, 29, 36, 43, 5Ø, 57, 64, 71)
(CTRL)T	Retypes the line currently being edited (in its present edited form)
(SHIFT)P	Clears screen of line currently being edited leaving the line in workspace as it was before calling it to be edited
(RUBOUT)	Deletes the character flashing with the cursor. Line closes up from the right.
EDITnn or !nn	Calls line number nn for editing
EDIT or !	Calls next line in program for editing
EDIT! or !!	Recalls last edited line for re-editing

iii) The TRAP command

An entirely new feature has been added to BASIC in V3.3. This feature, called TRAP, works much like "ON ERROR GOTO" in some other BASIC interpreters. As the name suggests, TRAP allows a BASIC program to retain control when an error in BASIC or DOS (Disk Operating System) occurs. To enable the TRAP feature, the command "TRAPnn" is used where nn is

a BASIC program statement number. If an error is encountered after the TRAP mode is enabled, the program will jump to the specified line number nn. This feature allows your program to make decisions on the basis of errors encountered while continuing to run. The TRAP mode is disabled by the command TRAPØ.

iv) New PRINT commands

Many new PRINT commands have been added to V3.3 BASIC which allow you to easily manipulate the screen display. The format of a displayed number can be specified. With one command, the cursor can be moved to any location on the screen (the "print at" feature). A wide character or narrow character screen format can be chosen. A display window on the screen can be specified. Single stepping of the cursor up and down as well as right and left is now possible. Line inserting and selective clearing of the screen can be done easily. Color manipulation with PRINT commands makes color displays easy to program. Cursor location and a character at the cursor location can be determined and used later in the program.

Additional PRINT statements will produce auto-paging on your printer and in the case of memory mapped video systems (but not serial systems) transfer the screen display to the printed page.

These new control features open up to you a whole new world of programming ease and flexibility.

Of course, the PRINT commands available in earlier versions of OS-65D are still present (see the OSI BASIC Reference Manual for a description of these). As was the case with the original commands (which include TAB(X), SPC(X), POS(X), and #M where M is a device number), several commands can be strung together in one PRINT statement. Examples of these combinations will be given as the commands are explained.

(a) Number Formatting

It is now possible to choose the format (number of digits and location of decimal point) for a number to be printed. The command is exemplified by PRINT USING "#.#". The character # represents a digit to be printed. A maximum of 11 '#' characters can be used if no decimal point is included. The maximum is 10 if a decimal point is included. No more than one decimal point can be used. Only a single format may be given with each PRINT USING statement. Example:

```
PRINT USING "###.##" 97.321, -1, 10000
```

will show on the screen:

```
 97.32 -1.00 ***.**
```

The "***.**" output indicates that the number was too large to fit the specified format.

The 'USING' designation may be combined with the '&' and '!' extensions described below, but must appear only once immediately after the word

PRINT and then will apply to the printing of all numerical values in that statement.

(b) Cursor Location ("print at")

The statement 'PRINT &(X,Y)' is the "print at" feature in V3.3 BASIC. It positions the cursor at X,Y on the screen. If a window smaller than the whole screen has been defined, the cursor goes to position x,y in the window. (Window definition will be explained later.) The origin of the X,Y coordinate system is the upper left corner of the screen or window. The X axis is horizontal and the Y axis is vertical, positive downward. After the cursor has been moved, the next character printed will be at the x,y position of the cursor.

WARNING: If a PRINT statement does not end with a semicolon, then a carriage return and line feed (CRLF) will automatically be performed. The CRLF action moves the cursor one line down and to the far left of the screen (or window). Usually you will not want this to happen because the next character printed would then appear at the left instead of the location x,y which you specified. Consequently, you should get in the habit of ending PRINT statements with a ";".

SECOND WARNING: Another condition can cause a surprise CRLF. If many "PRINT S;" (S is a set of expressions and/or commands) statements are used in succession, the line buffer (132 characters long) may fill up and a CRLF will be performed. The line buffer can be emptied either by a PRINT statement not ending in a semicolon or a comma (the comma specifies next zone on the same line) or by putting the expression CHR$(13); at the end of a PRINT statement. The first means of emptying the line buffer will produce a CRLF. The second technique will perform only a carriage return (cursor moved to the front of the line). In either case, you may need to reposition the cursor.

(c) General Screen Formatting

The video screen display can be controlled using any one or a combination of 25 commands as a part of a PRINT statement. Nineteen of the commands have the general form of PRINT!(XXX) where XXX is a set of one, two or three numbers or variables. Five of the commands have the form PRINT CHR$(nn) where nn is a decimal ASCII code. The one exception to these two forms is the PRINT &(X,Y) command described previously.

Of these 25 commands, three select the display size, 1Ø control the cursor location, one inserts a line, four clear part or all of the screen, five control color, and two will pick up information from the screen.

Display size PRINT commands

Of the three display size PRINT commands, two affect the entire screen and one sets a window. For total screen display, you can choose between a narrow character or wide character display. The narrow character display is a 32 line by 64 character format on the C4P and C8P while being 12 lines by 48 characters on the C1P. The wide character display is a 32x32 format on the C4P and C8P while being 24x24 on the C1P. Changing from one format to the other will clear the screen. For either format, a window can be set within which subsequent displays will be confined. Printing and clearing can be done within the window without affecting a display outside the window. However, the display outside the window cannot be selectively changed or cleared through use of PRINT commands.

The screen formatting PRINT commands are listed below:

PRINT!(2Ø)	Selects "wide letter" display (32x32 on C4P and C8P, 12x24 on C1P), clears the screen, and homes the cursor to upper left screen corner.
PRINT!(21)	Selects "narrow letter" display (32x64 on C4P and C8P, 24x48 on C1P), clears the screen, and homes the cursor to upper left screen corner.
PRINT!(22,w,h)	Selects print window w characters wide and h characters high. Upper left window corner is at current cursor position; screen is not cleared.

The ability to define a window makes for versatile formatting of text-on-text or text-on-graphics. However, there are certain conditions which must be kept in mind. There can be only one window defined at any given time. Once the window is defined, the cursor must remain within it, and so new text can only be written within the window. Text or graphics that were outside the window when it was formed will remain displayed. One may add text or change the graphics on the screen outside the window only by use of direct POKEs to the screen memory. The window may be redefined but only to a smaller window which must be entirely enclosed within the original window. Finally, one can escape the confines of the window only be executing either of the commands PRINT!(2Ø) or PRINT!(21) which will clear the screen and return the display to full screen size.

The !(22,w,h) window setting command is commonly used in the same PRINT statement with the &(X,Y) cursor positioning command which is used to determine the upper left corner of the window. If, however, the window setting command is in a separate PRINT statement, a ";" must follow the cursor positioning command. That is

```
1ØØ PRINT &(5,5)!(22,2Ø,1Ø);
```

and

```
1ØØ PRINT &(5,5);
11Ø PRINT !(22,2Ø,1Ø);
```

will have the same effect in setting a 1Ø line by 2Ø character window whose upper left corner is on the sixth line down and the sixth space from the left of the screen (x,y begin at Ø,Ø). Note that a ";" is used after the window setting command in order to leave the cursor in the upper left corner of the window.

Cursor location PRINT commands

Of the 1Ø cursor control PRINT commands, five will single-step the cursor to a new location, two will multistep the cursor, two allow the program to designate any location within the window, and one homes the cursor (returns it to the upper left corner of the window). The single step commands will move the cursor up and down as well as left and right. The The multistep commands translate the cursor only horizontally. In no case will characters on the screen be cleared when the cursor is moved by one of these commands.

The following is a list of the cursor control PRINT commands:

Single Step

```
PRINT CHR$(18)   Back one space
PRINT CHR$(16)   Forward one space
PRINT!(12)       Up one space
PRINT!(11)       Down one space
PRINT CHR$(1Ø)   Down one space
```

Multistep

PRINT CHR$(13)	Back to front of line (carriage return)
PRINT CHR$(14)	Forward to next eight space tab set (seven space for left-most field)

Anywhere

PRINT!(17,X,Y)	Relocate to X,Y (Ø,Ø at upper left corner)
PRINT &(X,Y)	Relocate to X,Y (Ø,Ø at upper left corner)

Home

PRINT!(18)	Relocates to Ø,Ø (upper left corner)

Note that PRINT!(11) and PRINT CHR$(1Ø) are identical commands and that PRINT!(17,X,Y) and RPINT &(X,Y) are also identical commands.

As indicated earlier, the commands listed above can be used together in one PRINT statement as well as in separate statements. For example, the statement

```
PRINT!(12)CHR$(16);
```

will move the cursor up one space then forward one space. Again, the ";" at the end of the statement prevents a CRLF from occurring.

Insert and clear PRINT commands

One PRINT command allows you to insert a line on the screen with a subsequent automatic moving down of the lower lines. The line will be inserted at the Y position of the cursor and begin at the location of the cursor.

The line insert command is listed below:

PRINT!(26) Inserts line at cursor position; lower lines scroll down

Four commands allow clearing of characters on the screen. It is important to remember that only the screen is affected by these commands. Workspace memory locations are not altered. Two of the commands affect only the line where the cursor is located when the command is executed, one clears a portion of the screen, and one clears the entire screen.

The clear PRINT commands are listed below:

Line

PRINT!(15) Clears from cursor to end of line
PRINT!(19) Clears entire line (lower lines move up)

Screen

PRINT!(24) Clears from cursor to end (lower right) of window
PRINT!(28) Clears entire screen and homes cursor in window

Note that the first three commands leave the cursor where it is (provided that the command is ended with a ";") while the last one returns the cursor to its home position. The home position is the upper left corner of the screen if the entire screen is being used, or is the upper left corner of the defined window. Note also that the first three commands affect only the window display region. Text displayed outside the window will not be cleared. However, PRINT!(28) clears the entire screen.

Color formatting PRINT commands

Sixteen different colors may be displayed by C1P, C4P and C8P computers. Each color is identified by a

4 bit number (the color value) which (in decimal form) identifies the color in BASIC programs. The colors are:

Color	Decimal Value
Yellow	Ø
Inverted Yellow	1
Red	2
Inverted Red	3
Green	4
Inverted Green	5
Olive Green	6
Inverted Olive Green	7
Blue	8
Inverted Blue	9
Purple	1Ø
Inverted Purple	11
Sky Blue	12
Inverted Sky Blue	13
Black	14
Inverted Black (white)	15

Notice that the colors are named in pairs. Bit zero of the binary form of the color value is one if a color has the word "inverted" in its name. This pairing of colors in an important concept in character display and in use of the color formatting PRINT commands.

Each character is made up of dots (pixels) in an 8x8 dot matrix "cell". The dots which produce the character are in one of the 16 colors, called the foreground color, while the rest of the dots in the cell are in another color, called the background color. You cannot pick the foreground and background colors independently from the list of 16 colors. The foreground color and the background color must be inverses

of each other. When a color number is chosen, the background appears in the color corresponding to that number while the foreground appears in the inverse color. For example, of color 2 is specified, the background will be red while the foreground will be in the color called inverted red. If color 3 is specified, the background will be inverted red while the foreground will be red. If you have a color monitor, run the program COLORS on Tutorial Disk 5 to observe these color combinations.

There are two types of color commands. One type specifies what color will be used henceforth as background color for characters displayed on the screen through the use of subsequent PRINT statements or keyboard entry (but not POKEs). Three commands are of this type. The other type affects all previously printed text of a specific color by either changing the color or clearing all text of that specific color. Two commands are of this type. These are powerful tools for generating and updating displays that can catch the attention of the viewer and transmit in a flash, through color coding, important information.

The five color formatting PRINT commands follow:

Color select

PRINT!(1)	Selects color Ø as the cell background
PRINT!(25)	Selects the normal black/white display mode (i.e., black selected as the cell background)
PRINT!(31,n)	Selects color n as the cell background

Color change

PRINT!(2,n,m)	Changes cell color. All cells on the screen of background color m are changed to background color n.
PRINT!(29,n)	Clears cells of background color n. All cells on the screen of background color n are changed to black background and the character is replaced with a blank

WARNING: The color formatting PRINT commands will not function until a screen format has been specified. A program which uses color formatting must begin with PRINT!(21) or PRINT!(2Ø) in order to run properly right after your computer has been turned on.

Cursor sensing PRINT commands

OS-65D V3.3 allows you to determine the position of the cursor while a program is running then use that information as a part of the program. Furthermore, you can pick up the character on which the cursor is located and store that character as a string variable.

The commands which allow this type of manipulation follows.

PRINT!(5)	Sends information for current cursor position X,Y to string variable in following INPUT statement. Information is in the form of two characters for which (X+65), (Y+65) is the ASCII code. Line feed follows the INPUT statement used with PRINT!(5)
PRINT!(33)	Sends character at cursor position to string variable in following INPUT statement. Line feed follows the INPUT statement used with PRINT!(33)

The cursor position sensing is done by way of ASCII characters which are recognized by the BASIC interpreter. Consequently, the ASC(X$) function is needed to obtain the numerical values of X and Y. For example, the program shown below will determine the position of the cursor after it has printed a line of text.

```
 1Ø REM CURSOR POSITION
 2Ø PRINT!(28)!(17,1Ø,1Ø);:REM CLEAR SCREEN AND POSITION CURSOR
 3Ø PRINT"CURSOR POSITION SENSING TEST.";
 4Ø PRINT!(5):REM PICK UP CURSOR POSITION INFORMATION
 5Ø INPUT P$:REM STORE INFORMATION IN P$
 6Ø PRINT:PRINT"ASCII CHARACTERS FOR X+65 AND Y+65 ARE ";P$
 7Ø X = ASC(P$) - 65:REM CONVERT ASCII CHARACTER TO X VALUE
 8Ø PRINT:PRINT"THE VALUE OF X AT END OF TEXT LINE IS";X
 9Ø Y$ = RIGHT$(P$,1)
1ØØ Y = ASC(Y$) - 65:REM CONVERT ASCII CHARACTER TO Y VALUE
11Ø PRINT"THE VALUE OF Y FOR THE TEXT LINE IS";Y
12Ø END
```

As written, the program will store "hK" in P$ and translate these letters into an X position of 39 and a Y position of 1Ø. If you were to change the 1Ø,1Ø in line 2Ø, and/or change the message of line 3Ø, a different pair of characters would be stored in P$ and a different set of X,Y values would be determined.

WARNING: There is a limitation to the use of PRINT!(5). The X values of 26, 3Ø, and 58 through 63 cannot be picked up. This is because this PRINT command does not recognize the ASCII characters corresponding to these values plus 65. An alternate technique which works for all X and Y values is the PEEKing of address 13Ø23 for X and 13Ø24 for Y. That is, replacing lines 4Ø through 7Ø with

```
4Ø X=PEEK(13Ø23)
```

and lines 1Ø and 1ØØ with

```
9Ø Y=PEEK(13Ø24)
```

will result in the same values of X and Y being determined.

A similar limitation holds for the character pick up command. Certain characters will not be recognized by CHR$, PRINT!, and other BASIC commands and functions. ASCII codes Ø through 6, 8 through 12, 14 through 31, 93, 94, 95, 125, 126, and 127 will not be properly recognized. For those characters that are recognized properly, the following program is useful:

```
 1Ø REM CHARACTER PICK-UP
 2Ø PRINT!(28)&(1Ø,1Ø);:REM CLEAR SCREEN AND POSITION CURSOR
 3Ø PRINT"ABCDEFGHIJKLMNOPQRSTUVWXYZ"
 4Ø PRINT:INPUT"ENTER A NUMBER 1 THROUGH 26";N
 5Ø PRINT!(17,9+N,1Ø);:REM SET CURSOR ON LETTER CHOSEN BY N
 6Ø FOR T = 1 TO 5ØØØ:NEXT:REM DELAY TO OBSERVE CURSOR LOCATION
 7Ø PRINT!(33):REM PICK UP LETTER
 8Ø INPUT CH$:REM STORE LETTER IN CH$
 9Ø PRINT$(Ø,5);
1ØØ PRINT"THE LETTER PICKED UP IS ";CH$
11Ø END
```

Note that the above program uses PRINT&(X,Y) instead of PRINT!(17,X,Y). As stated earlier, these commands are completely interchangeable.

(d) Printer Control

The OS-65D printer drivers will now perform auto-paging by recognizing a "top-of-form" command. To initialize this feature, execute this command in the immediate mode (printer on):

```
PRINT#M,!(67,FL)
```

where M is the printer device number (1 for serial printers) and FL is the form length (number of lines per page). For a form 11 inches long and a printer set at six lines per inch, FL=66. Immediately upon receiving the top of form command, the printer will advance the appropriate number of "between form" lines (defined below) and stop. Then position your form in the printer so the top of the form is under the print head. You are now ready to use the system. As you run your programs, the system will count lines and automatically advance to the next page when the bottom of the form is reached. The number of lines it inserts between the bottom of one page and the top of the next (the "between form" lines) is FL/1Ø, truncated to an integer. So in our example where FL=66, there are six lines between pages (three line top and bottom margins on each page).

Normally, of course, a printout will end in the middle of a page. To begin a new printout at the top of a page, use the command

```
PRINT#M,CHR$(12)
```

This command sends enough line feeds to position the printer at the top of the next page.

If you have a memory mapped video system (not serial with a video terminal) and an Epson MX-80 printer, you can send text and 16 medium resolution (64x128) graphics characters to the printer. The BASIC command is PRINT#1,!(80). This command will print what is on the screen (or in the window) and will put a black boarder around it. However, only standard text and 16 medium resolution graphics characters are recognized by the printer. The standard text includes the keyboard characters and OSI character graphics 33 through 126. The 16 graphics characters are numbers 32, 154, 157, 165, 166, 167, 168, and 170 from the OSI graphics character set and their inverses. An inverse is formed by POKEing the corresponding color cell with an inverse color number. For black and white displays, the inverse color number to use is 15. Note that color does not need to be turned on for the character inverse to be formed. A sample program which will print the 16 graphics characters on an Epson MX-80 printer follows:

```
  5 REM EXAMPLE OF GRAPHICS CHARACTERS PRINTING
 1Ø PRINT!(28):REM CLEAR SCREEN
 2Ø PRINT&(Ø,1Ø)"TEST OF PRINTER AND GRAPHICS"&(Ø,14);
 4Ø GS = 54272:REM START OF GRAPHIC CHARACTER PRINT
 5Ø IS = 54274 + 4096:REM START OF INVERSE CHARACTER PRINT
 6Ø FOR G = Ø TO 28 STEP 4
 7Ø READ GC:REM READ CHARACTER NUMBER FROM DATA
 8Ø POKE GS+G,GC:REM DISPLAY CHARACTER
 9Ø POKE IS+G,15:POKE GS+G+2,GC:REM DISPLAY CHARACTER INVERSE
1ØØ NEXT
11Ø PRINT"TEST IS COMPLETED
12Ø PRINT#1,!(8Ø):REM SEND SCREEN TO PRINTER
2ØØ DATA32,154,157,165,166,167,168,17Ø
21Ø END
```

When the print routine does not recognize a character on the screen, the following translation between screen character and printed character is made. If the corresponding color cell does not have the inverse bit (bitØ) set (i.e., if the color number is even) then a blank is printed. If the corresponding color cell does have the inverse bit set (i.e., if the color number is odd) then a solid black square is printed.

Now that the full range of new PRINT commands has been described, the examination of a few programs will help to illustrate their use. The programs shown below combine many of the commands in one PRINT statement as well as illustrate the ease and flexibility of programming with the PRINT commands.

The first program listed below is designed to run on a color monitor, but will be illustrative of programming capabilities on a black and white monitor.

This program can be terminated only by using CTRL-C. (Note: Serial system users should consult the NOTE at the end of this chapter.)

```
 1Ø REM *** "PRINT AT" DEMO. ***
 2Ø PRINT!(28)!(21):REM CLEAR SCREEN:ENABLE COLOR
 3Ø M$ = "HELLO, I'M HERE NOW!"
 4Ø Y = RND(1)*23:REM GENERATE Y POSITION
 5Ø X = RND(1)*42:REM GENERATE X POSITION
 6Ø C = INT(RND(1)*15):REM GENERATE COLOR NUMBER
 7Ø POKE 2Ø73,96:POKE 13Ø26,32:REM DISABLE CTRL C:BLANK CURSOR
 8Ø PRINT&(X,Y)!(31,C)M$CHR$(13)!(18);
 9Ø FOR T = 1 TO 3ØØØ:NEXT:REM TIME DELAY
1ØØ POKE 13Ø26,171:POKE 2Ø73,173:REM REST. CURS.:ENABLE CTRL C
11Ø GOTO 3Ø
```

Note that the program above illustrates the "print at" capability of V3.3. It also shows the ease with which color can be included. An additional feature of this program is the removal of the cursor (by replacing the square with a blank cell) while the program runs. If, however, a program in which the cursor has been removed is interrupted before the cursor is replaced, the cursor will remain invisible. Consequently, CTRL-C is disabled while the cursor is blanked out.

The second program listed below shows the ease with which displays can be sorted into distinct parts. It is designed for a black and white monitor. It illustrates also a use of the character pickup feature. (Note: Serial systems users should consult the NOTE at the end of this chapter.)

```
 1Ø REM NUMBER SORT DEMONSTRATION
 2Ø POKE 13Ø26,32:REM BLANK CURSOR
 3Ø PRINT!(28)!(2Ø)&(4,8)"UNDER 5Ø"!(31,15)&(2Ø,8)"OVER 5Ø"
 35 PRINT&(Ø,6)"Random Number Sort Demonstration"
 4Ø READ P$(1),P$(2):REM READ MESSAGES FROM DATA
 5Ø FOR I = 1 TO 1Ø
 6Ø Y = 1Ø
 7Ø N = RND(1)*1ØØ
 8Ø IF N <=5Ø THEN X = 4:C = 14:GOTO 1ØØ
 9Ø X = 2Ø:C = 15
1ØØ GOSUB 2ØØ
11Ø PRINT USING"##.##"&(X,Y)!(31,C)N
12Ø GOSUB 3ØØ
13Ø NEXT
14Ø PRINT!(31,14)!(18):REM SET BLK BKGD, HOME CURSOR
15Ø POKE 13Ø26,171:REM REPLACE CURSOR
16Ø END
2ØØ REM SUBROUTINE TO SELECT Y VALUE
2Ø5 PRINT&(X+3,Y)!(33)
21Ø INPUT CH$
22Ø IF CH$ = "." THEN Y = Y+1:GOTO 2ØØ
23Ø RETURN
3ØØ REM SUBROUTINE TO FLASH MESSAGE
3Ø5 FOR M = 1 TO 5
31Ø IF C = 14 THEN C = 15:D = 1:GOTO 34Ø
32Ø C = 14
33Ø D = 2
34Ø PRINT&(2,2Ø)!(31,C)P$(D)
35Ø FOR T = 1 TO 5ØØ:NEXT T
36Ø NEXT M
37Ø RETURN
5ØØ DATA"NOTE RANDOM NUMBER SELECTION"
51Ø DATA"note random number selection"
```

Note that the subroutine beginning at line 2ØØ checks to see whether the last decimal point is in either the left or right (low number or high number) column of listed numbers. When the decimal point is found to not be present, the number is printed. Of course, the program would run in an instant if line 12Ø (the call of the message flashing subroutine) were removed. It is included to illustrate the ease with which attention getting displays can be programmed.

Line insert and line clear capabilities are illustrated in the program below. (Note: Serial system users should consult the NOTE at the end of this chapter.)

```
 1Ø REM LINE INSERT DEMONSTRATION
 2Ø PRINT!(28)!(21);:REM CLEAR SCREEN, HOME CURSOR, SET FORMAT
 3Ø m1$ = "Original line"
 4Ø m2$ = "of text is here."
 5Ø FOR L = 1 TO 2Ø
 6Ø PRINTm1$;L;m2$
 7Ø NEXT
 8Ø PRINT&(Ø,21)"Enter line to be inserted (1 - 61 char.)."
 9Ø INPUT L$
1ØØ PRINT!(12)!(19)!(12)!(19);:REM CLEAR MESSAGE LINES
11Ø INPUT"Enter number of line where insert is to go.";N
12Ø PRINT!(12)!(19)&(Ø,N-1)!(26)L$&(Ø,21);
13Ø END
```

Lines 3Ø through 7Ø produce text on the screen. Lines 8Ø through 11Ø ask you to enter a line of text and then a location for it to be inserted. But before the insert location is asked for, the first message and your entered text are cleared. Line 12Ø

clears the message, inserts the text you have entered, then moves the cursor to the lower left of the screen so that the OK prompt will appear there when the program ends.

v) Data file handling

The BASIC data file handling of V3.3 has also been modified. First, execution speed has been improved for the DISK GET command and the DISK PUT command is no longer necessary. Second, a DISK FIND command has been added to allow rapid searching for a string of characters within a BASIC data file. The syntax for the command is:

```
DISK FIND, string
```

Examples of the FIND command are

```
DISK FIND, "HARRY"
```

and

```
DISK FIND, AB$
```

The command works most smoothly with sequential data files. The search begins at the current file pointer location. If you wish to place the pointer at the beginning of the file, you must execute DISK OPEN,n,"filename" prior to the DISK FIND command. (n is buffer number 6 or 7.) If the string is located in the data file, the pointer will be left at the end of the field in which the string was found. If the string is not found, a disk error #D will be reported. The TRAP command mentioned earlier is useful for recovering from the error without exiting the BASIC program.

Use of the FIND command with random access files is possible if the programmer has been careful in filling interrecord gaps with nulls, spaces or carriage returns. That is, a 128 byte record which at one point had data 100 bytes long and then was re-written with 50 byte data will still contain 50 bytes of the old data. The FIND command may find this old data instead, but the information returned by way of the following INPUT statement will be the new data. If spaces are used for padding, they should precede the data. If carriage returns are used, they should come after the data. Nulls (character for ASCII code 0) can be used either before or after the data.

Since the FIND command will usually terminate with the file pointer set inside a record, a BASIC subroutine has been included which will calculate the number of the record in which the FIND command terminated. The subroutine is listed below.

```
900 REM Record calculation subroutine for DISK FIND
910 DEF FN dec(bcd) = (bcd AND 15) + (INT(bcd/16)*10)
920 :
930 ab=PEEK(12042) * (FN dec(PEEK(9004)) - FN dec(PEEK(9002)))
940 re=(PEEK(9133)-PEEK(8999))*256 + PEEK(9132)-PEEK(8998)-1
950 rn=ab + INT(re/2^PEEK(12076))
960 :
970 REM rn is the desired record number
980 RETURN
```

On a random access data file, the DISK FIND command should be followed by GOSUB 900:DISK GET,RN where RN is the desired record number. Then the DISK GET command can be used to set the file pointer to the beginning of the record that contained the search string.

The program on the following page is an example of the use of the DISK FIND command for a sequential file. It is intended for use with Tutorial Disk 4 and, if you saved the sequential file demonstration program given in Chapter 4, it can be entered by adding to that program.

Boot up your computer with Tutorial Disk 4, then choose option 1. If the earlier example was saved, you will now have about half of the DISK FIND demonstration program in the workspace. Complete the program by changing line 3Ø then adding lines 7Ø, 213, 255, and 3ØØ through 53Ø. If you did not save the earlier example, enter NEW then type in the entire program. When completed, the program should be saved in PROG1.

Note that the program now includes the TRAP feature which is used in both the listing routine and the finding routine. Care has been taken to disable the TRAP feature after each intentional use so that an error #D message occurring some other place in the program will indicate an actual error.

When you have stored three sets of names and numbers, choose the "Find a name" option. Next, enter one of the names you have stored. The program will find that name, then pickup the corresponding number (the next item in the sequential file). The number must be separated from the name by a carriage return (i.e., entered into the file by a separate PRINT statement as done in line 14Ø). The

```
10 REM SEQUENTIAL FILE DISK FIND DEMONSTRATION
20 PRINT!(28)
30 INPUT"Enter data, List data, or Find a name (E/L/F)";C$
40 PRINT!(28)
50 IF C$ = "E" THEN 100
60 IF C$ = "L" THEN 200
70 IF C$ = "F" THEN 300
100 REM ROUTINE FOR ENTERING DATA
105 DISK OPEN,6,"DATA1"
110 FOR I = 1 TO 3
120 INPUT"Enter a name";N$
130 INPUT"Enter the telephone number";T$
140 PRINT#6,N$:PRINT#6,T$
150 NEXT I
160 DISK CLOSE,6
170 PRINT"DATA STORED."
180 GOTO 30
200 REM ROUTINE FOR LISTING DATA
210 DISK OPEN,6,"DATA1"
215 TRAP 400
220 FOR I = 1 TO 3
230 INPUT#6,N$,T$
240 PRINT N$,T$
250 NEXT I
255 TRAP 0
260 DISK CLOSE,6
270 GOTO 30
300 REM ROUTINE FOR FINDING DATA
310 INPUT"Enter the name to be found";N$
320 DISK OPEN,6,"DATA1"
330 TRAP 500
340 DISK FIND,N$
345 TRAP 0
350 INPUT #6,T$
360 PRINT N$,T$
365 INPUT"Continue search (Y/N)";A$
370 IF A$ = "Y" THEN 330
380 DISK CLOSE,6
390 GOTO 30
400 REM LIST ROUTINE TRAP MESSAGE
410 PRINT"THE DATA FILE IS EMPTY."
420 TRAP 0
430 GOTO 30
500 REM FIND ROUTINE TRAP MESSAGE
510 PRINT"THAT ENTRY WAS NOT FOUND."
520 TRAP 0
530 GOTO 30
```

PRINT statement on line 36Ø prints the name you have entered and the corresponding number. If the name is not found, the message in line 51Ø will be displayed.

Note that you can find the name by entering any sequential segment of the spelling of the name. The screen display, however, will give the partial spelling, as you entered it, not the name as it is stored in the the file. In a sequential file, the pointer cannot back up to allow retrieval of the item found. This could give a problem if, for example, two people with the same last name but different first names were on the list. A search for the last name would return the telephone number for the first person with that name on the file but would not return the first name. A continuation of the search would give only a second number. If you had temporarily forgotten the first name, you could not get the correct number without starting from the beginning of the sequential file and listing the entries up through the two names. The next example shows how the FIND command can be used with random files in order to return the entire record in which the found item resides.

The program shown next can be formed by adding to the random file demonstration program in Chapter 4. Boot up your computer with Tutorial Disk 4, then choose option 2 (or simply choose option 2 from the menu if you are already booted up with disk 4). If the earlier example was

```
10 REM RANDOM FILE DISK FIND DEMONSTRATION
20 PRINT!(28)
30 INPUT"Enter record, List record, or Find item (E/L/F)";C$
40 PRINT!(28)
50 IF C$ = "E" THEN 100
60 IF C$ = "L" THEN 300
70 IF C$ = "F" THEN 500
100 REM ROUTINE FOR ENTERING DATA
110 DISK OPEN,6,"DATA2"
120 FOR R = 1 TO 3
130 PRINT"For record number";R;
140 INPUT"enter item name";I$
150 INPUT"Enter number of items";N$
160 DISK GET,R
170 PRINT#6,R;",";I$;",";N$
180 DISK PUT
190 NEXT R
200 DISK CLOSE,6
210 PRINT"DATA STORED"
220 GOTO30
300 REM ROUTINE FOR LISTING RECORDS
310 INPUT"Enter number for record to be listed";R
320 DISK OPEN,6,"DATA2"
325 TRAP 700
330 DISK GET,R
340 INPUT#6,RN,I$,N$
345 TRAP 0
350 PRINT RN,I$,N$
360 INPUT"List another record (Y/N)";A$
370 IF A$ = "Y" THEN 310
380 DISK CLOSE,6
390 GOTO 30
500 REM ROUTINE FOR FINDING ITEMS
510 INPUT"Enter the item to be found";I$
520 DISK OPEN,6,"DATA2"
530 TRAP 800:DISK FIND,I$:TRAP0
540 GOSUB 900
550 DISK GET,RN
560 INPUT#6,RN,IR$,N$
570 PRINT RN,IR$,N$
580 DISK CLOSE,6
590 GOTO 30
700 REM LIST ROUTINE TRAP MESSAGES
710 IF R < 4 AND R > 0 THEN PRINT"FILE IS ERASED.":GOTO 730
720 PRINT"ONLY RECORDS 1, 2, AND 3 ARE IN THE FILE."
730 TRAP 0
740 GOTO 30
800 REM FIND ROUTINE TRAP MESSAGE
810 PRINT"THAT ITEM IS NOT IN THE FILE."
820 TRAP 0
830 GOTO30
900 REM Record calculation subroutine for DISK FIND
910 DEF FN dec(bcd) = (bcd AND 15) + (INT(bcd/16)*10)
930 ab=PEEK(12042) * (FN dec(PEEK(9004)) - FN dec(PEEK(9002)))
940 re=(PEEK(9133)-PEEK(8999))*256 + PEEK(9132)-PEEK(8998)-1
950 rn=ab + INT(re/2^PEEK(12076))
970 REM rn is the desired record number
980 RETURN
```

saved, you will now have nearly half of the random file DISK FIND demonstration program in workspace. Complete the program by changing line 3Ø then adding lines 7Ø, 325, 345, and 5ØØ through 98Ø. If you did not save the earlier example, enter NEW then type in the entire program. When completed, be sure to save the program in PROG2.

Before running the program, choose option 5 from the Tutorial Disk 4 menu to erase data file DATA2. If you do not do this, you may get some surprises when using the FIND option (because of data left in a record from previous use).

Now choose option 2 from the menu and run the demonstration program. After storing data, try finding an item. You can use all or a sequential portion of the letters in the item name. The FIND routine will return the entire record as it is stored in the file. As in the earier DISK FIND demonstration program, the TRAP feature will return a message if the item is not found.

D. BASIC Functions not present in OS-65D V3.3

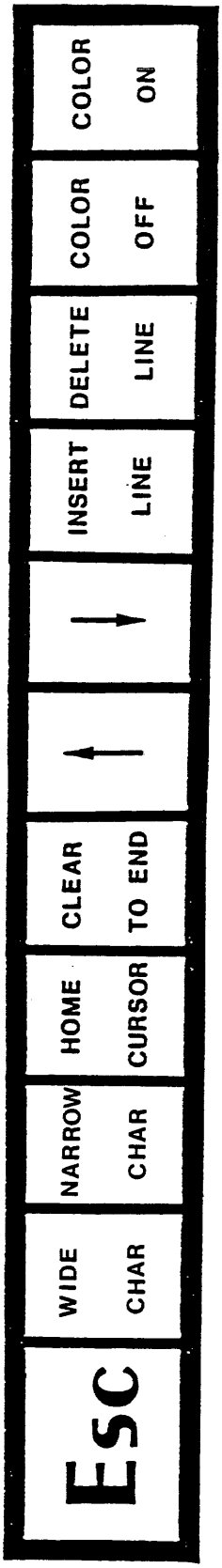
There are a few BASIC functions in OS-65D V3.2 that are no longer present in V3.3. The first two of these functions are NULL and WAIT. These keywords have been replaced by EDIT and TRAP. The third function that has been disabled is ATN (arctan).

However, since this keyword has not been reused, it is possible to revive this function if it is crucial to your applications. The cost of reviving ATN is the loss of the PRINT extensions just described. If you wish to use ATN, run the program called 'ATNENB' contained on disk 5 of the tutorial set. A menu will be printed giving you a choice between the ATN function or the PRINT extensions. After you select which feature you want, the program will automatically reconfigure your system to support either ATN or the PRINT extensions.

OVERLAY CUTOUT

For Screen Display Commands

Users of Video Systems may find the overlay on this page helpful for remembering the ESC key screen display commands. It should be cut out and placed above the top row of keys so that the "WIDE CHAR" is just above the "1" key.



** Note to Users of Serial Systems **

OS-65D V3.3 is only partially compatible with serial systems. If you are using a Hazeltine 142Ø terminal, be sure switch 6 is set to the ESC position. Certain features that refer to color, screen size, or windowing are not operable on serial systems. Specifically,

1) The commands that use the ESC key are not operable.

2) The destructive backspace key is <DEL> instead of <SHIFT/O> or <RUB OUT> and the line delete is <@> instead of <SHIFT/P>.

3) The PRINT command !(26) inserts a line but not at the cursor position. The line always starts at the left margin.

4) The following PRINT commands should not be used:

!(1)	Color Ø select
!(2,n,m)	Color change
!(5)	Cursor position sensing
!(2Ø)	Wide character select
!(21)	Narrow character select
!(22,w,l)	Window select
!(25)	Color black/white select
!(28)	Screen clear
!(29,n)	Selective color clear
!(31,n)	Color n select
!(33)	Character pick up

Chapter 8

Extended 65D Utilities

When Tutorial Disk Two is booted up no menu is displayed. Instead, the system is unlocked for programming in BASIC's immediate mode (the "OK" prompt appears). What is not readily apparent is that this disk also contains a package of powerful 65D utility programs. With these programs, a 65D user can:

1. Change the numbering of statements within a program in a variety of ways. This makes it possible, for example, to add a statement between two statements that presently are numbered consecutively.

2. Reduce the storage space required by a program by removing REM statements and/or blanks.

3. Check the size of buffers present before a program. Buffers can also be added and deleted in a very flexible fashion.

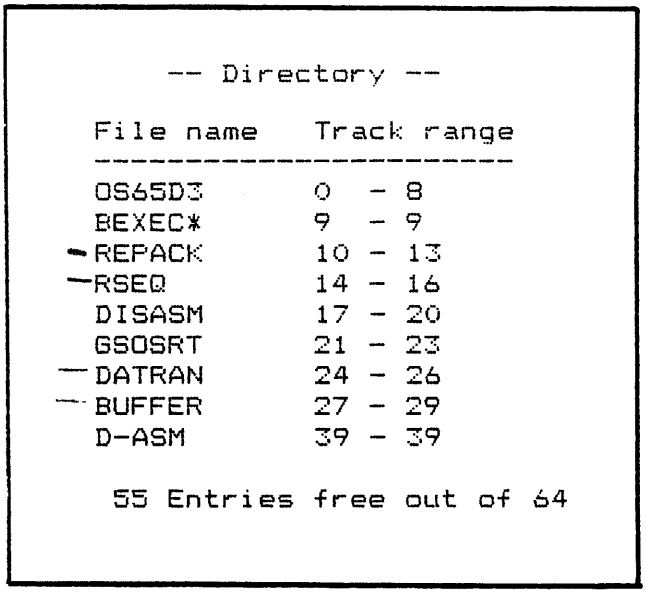
4. Sort data files, including MDMS master files.

5. Generate an assembly language listing for machine code programs.

6. Copy data files.

These utility programs can process program and data files created under any version of OS-65D. However, several of the utility programs must be run on a system booted under 65D V3.2. For this reason, Tutorial Disk Two boots under 65D V3.2 This means that programming done after booting the system with Tutorial Disk Two cannot use the new features of 65D V3.3 that are available when the system is booted with Tutorial Disks three, four, or five.

The directory listing for Tutorial Disk Two, C4P 5" system is:



```
-- Directory --

File name    Track range
-----------------------
OS65D3       0  - 8
BEXEC*       9  - 9
REPACK       10 - 13
RSEQ         14 - 16
DISASM       17 - 20
GSOSRT       21 - 23
DATRAN       24 - 26
BUFFER       27 - 29
D-ASM        39 - 39

 55 Entries free out of 64
```

(track ranges for 8" systems will be somewhat different).

Disk Two is write-protected to reduce the possibility of accidentally destroying these files.

Detailed explanations and examples for the Extended 65D Utility programs follow. Listings for all of the programs appear at the end of this chapter.

A. Resequencer

The resequencer is stored in the file RSEQ. To use this program, boot the system with Tutorial Disk Two and type RUN"RSEQ. Then type "E" or "ENABLE" in response to the enable/disable question. This replaces the keyword "NULL" with the word "RSEQ" and places code to implement the RSEQ command at the top of memory. RSEQ cannot be run on a system booted under 3.3 because the 3.3

Editor uses the top of memory. With RSEQ enabled, any 65D program can be loaded into the workspace (don't reboot - simply insert the disk containing the PROGram and type DISK!"LO PROG") and resequenced in a variety of ways.

The syntax for the RSEQ command is as follows:

NLN = new line number Ø<=NLN<64ØØØ

OLN = old line number Ø<=OLN<64ØØØ

INC = increment between line numbers Ø<INC<256

RSEQ<CR>	- resequence starting with the line number 1Ø at the first line and renumber the lines in increments of ten.
RSEQ NLN<CR>	- resequence using NLN as the first line number and renumber the lines that follow by increments of ten.
RSEQ NLN,OLN<CR>	- resequence starting at line OLN with line number NLN and renumber the lines that follow by increments of ten.
RSEQ NLN,OLN,INC<CR>	- resequence starting at line OLN with the line number NLN and renumber the lines that follow by increments of INC.
RSEQ NLN,,INC<CR>	- resequence starting with the first line as NLN and renumber the lines that follow by increments of INC.
RSEQ ,,INC<CR>	- resequence starting with the first line and renumber the lines that follow by increments of INC.

If a non-existent line is referred to, an undefined statement error will appear with the old line number, the new line number and the line being resequenced. After the resequence is complete, the next available line number in the sequence will be output between square brackets.

The resequencer will prevent the buffer creator from being enabled when it is enabled. The resequencer can be disabled by running "RSEQ" and typing "D" or "DISABLE" to the enable/disable question.

Examples: Consider the following program:

```
10 REM TEST PROGRAM
20 REM
30 PRINT:PRINT"THIS IS THE TEST PROGRAM":REM
40 X = 0
50 REM:REM
60 X = X + 1
70 ON X GOTO 100, 200, 300: LET X = 1000
80 PRINT:PRINT:PRINTX:PRINT"THIS IS THE END"
90 END
100 PRINT:PRINT"AT STATEMENT 100": GOTO 60
200 PRINT:PRINT"AT STATEMENT 200": GOTO 60
300 PRINT:PRINT"AT STATEMENT 300": GOTO 60
```

The command RSEQ will resequence the entire program by line number increments of 10, starting with a line number of 10. The result is:

```
10 REM TEST PROGRAM
20 REM
30 PRINT:PRINT"THIS IS THE TEST PROGRAM":REM
40 X = 0
50 REM:REM
60 X = X + 1
70 ON X GOTO 100, 110, 120: LET X = 1000
80 PRINT:PRINT:PRINTX:PRINT"THIS IS THE END"
90 END
100 PRINT:PRINT"AT STATEMENT 100": GOTO 60
110 PRINT:PRINT"AT STATEMENT 200": GOTO 60
120 PRINT:PRINT"AT STATEMENT 300": GOTO 60
```

Notice that the line numbers referred to in line 70 are correctly changed. However, statement numbers that appear in PRINT strings (enclosed in quotes) do not change.

The command RSEQ 500 also resequences the entire program with increments of 10, but starts with a line number of 500 for the first line. The result is:

```
500 REM TEST PROGRAM
510 REM
520 PRINT:PRINT"THIS IS THE TEST PROGRAM":REM
530 X = 0
540 REM:REM
550 X = X + 1
560 ON X GOTO 590, 600, 610: LET X = 1000
570 PRINT:PRINT:PRINTX:PRINT"THIS IS THE END"
580 END
590 PRINT:PRINT"AT STATEMENT 100": GOTO 550
600 PRINT:PRINT"AT STATEMENT 200": GOTO 550
610 PRINT:PRINT"AT STATEMENT 300": GOTO 550
```

Finally, the command RSEQ 99,50,100 resequences with increments of 100, starting with a line number of 99 at old line 50. The result is:

```
10 REM TEST PROGRAM
20 REM
30 PRINT:PRINT"THIS IS THE TEST PROGRAM":REM
40 X = 0
99 REM:REM
199 X = X + 1
299 ON X GOTO 599, 699, 799: LET X = 1000
399 PRINT:PRINT:PRINTX:PRINT"THIS IS THE END"
499 END
599 PRINT:PRINT"AT STATEMENT 100": GOTO 199
699 PRINT:PRINT"AT STATEMENT 200": GOTO 199
799 PRINT:PRINT"AT STATEMENT 300": GOTO 199
```

B. Repacker

The REPACK Utility allows users of OS-65D to pack their programs into the smallest possible amount of space. This is done by removing REM statements and/or blank spaces. Before REPACKing, the program must be saved in an OS-65D named file. Once this has been done, reboot with Tutorial Disk Two and type RUN"REPACK. Insert the disk which contains the program you wish to REPACK into Drive A and enter the program's file name. Now enter the method of REPACKing (remove blanks, REM's or both). Enter 1, 2, or 3. When the operations are completed, a message will appear reporting the number of bytes removed from the program.

Note that this program can be run only with the system booted under OS-65D V3.2 (Tutorial Disk Two).

Examples: Consider the following program:

```
10 REM TEST PROGRAM
20 REM
30 PRINT:PRINT"THIS IS THE TEST PROGRAM":REM
40 X = 0
50 REM:REM
60 X = X + 1
70 ON X GOTO 100, 200, 300: LET X = 1000
80 PRINT:PRINT:PRINTX:PRINT"THIS IS THE END"
90 END
100 PRINT:PRINT"AT STATEMENT 100": GOTO 60
200 PRINT:PRINT"AT STATEMENT 200": GOTO 60
300 PRINT:PRINT"AT STATEMENT 300": GOTO 60
```

Running REPACK results in the following screen display:

```
OK
RUN"REPACK



** BASIC PROGRAM REPACK UTILITY **

** WARNING --- USE DRIVE A ONLY **



FILENAME? TEST

REPACK BY REMOVING:
 1> BLANK SPACES
 2> REM STATEMENTS
 3> BOTH BLANK SPACES AND REMS
ENTER YOUR CHOICE? 3

 40 BYTES RECOVERED.

OK
```

Then LISTing the program produces the following:

```
10 REM
20 REM
30 PRINT:PRINT"THIS IS THE TEST PROGRAM"
40 X=0
50 REM
60 X=X+1
70 ONXGOTO100,200,300:LETX=1000
80 PRINT:PRINT:PRINTX:PRINT"THIS IS THE END"
90 END
100 PRINT:PRINT"AT STATEMENT 100":GOTO60
200 PRINT:PRINT"AT STATEMENT 200":GOTO60
300 PRINT:PRINT"AT STATEMENT 300":GOTO60
```

Notice that 40 bytes are saved in the second program. This is done in the following way:

line	# bytes saved	
10	13	(13 character REMark)
30	2	(":" and keyword "REM")
40	2	(2 spaces)
50	4	(4 character REMark)
60	4	(4 spaces)
70	9	(9 spaces)
100	2	(2 spaces)
200	2	(2 spaces)
300	2	(2 spaces)
Total:	40	

Note that no lines are removed, even if they contain only the keyword "REM".

C. Buffer Creator

The Buffer Creator also utilizes the top of memory. Hence, it cannot be run on a system booted under 65D V3.3, or a system with the RSEQ command enabled. The program is stored in the file BUFFER.

To use it, boot the system with Tutorial Disk Two (65D V3.2) and type RUN"BUFFER. Respond "E" or "ENABLE" to the enable/disable question. The keyword "NULL" will be replaced with the word "BYTE" and code will be placed at the top of memory that allows use of the BYTE command. Then load the program for which you wish to create (or delete) buffers and use the BYTE command according to the following syntax:

Ø=<NB<64ØØØ

BYTE NB<CR>- moves source leaving NB free bytes in between the operating system and the source.

BYTE<CR> - reports the start of the buffer, the starting byte of your program and the number of bytes between the source and the operating system.

Examples: Consider a program with no buffer, i.e., it starts at the beginning of the workspace. The command BYTE will produce the following listing:

```
OK
BYTE
                        DECIMAL   HEXADECIMAL
 BUFFER  STARTS :        12670       $317E
 PROGRAM STARTS :        12670       $317E
 BUFFER   SIZE  :        Ø           $0000

OK
```

The command BYTE 6144 would be appropriate for saving two single-track (3Ø72 byte) buffers for, say, data file use with this program on an 8" system. The following listing is produced:

```
OK
BYTE 6144
                        DECIMAL   HEXADECIMAL
 BUFFER  STARTS :        12670       $317E
 PROGRAM STARTS :        18814       $497E
 BUFFER   SIZE  :        6144        $1800

OK
```

On a 5" system, BYTE 4Ø96 would create two single-track buffers.

To remove buffers from a program, simply enter the command BYTE Ø (the command BYTE is not the same as BYTE Ø; BYTE simply reports information. It does not alter existing buffers in any way).

With the BYTE NB command, all locations in the buffer are set to zero; therefore, anything of value in the buffer area should be saved before executing the command. After buffers are created they can be saved on disk by saving the program.

When BYTE is enabled, the RSEQ command cannot also be enabled. To disable BYTE, simply type RUN"BUFFER and respond "D" or "DISABLE".

D. General String Oriented Sort

GSOSRT is a general purpose sorting program that may be utilized on virtually any data file under OS-65D. The program allows for multiple fields in either sequential or random format and has special conditions to accommodate MDMS Master Files.

To use GSOSRT on a MDMS master enter the master file name (the "Ø" extension is optional) and type a "Y" or "YES" to the question "IS THIS A MDMS MASTERFILE?". The program will then list the fields that are present in the file. Select, by number, the field you want the file to be sorted by. The program will then list your responses up to this point and request the user's final confirmation. A response of "Y" or "YES" will initiate the sort. Any other response will abort the program.

To use GSOSRT with a non-MDMS type file, first specify whether the file is sequential or random. If a random file is specified, the user must additionally specify the number of bytes per record. Default record size is 128 bytes under OS-65D. The program will then ask for the number of fields per record, and the number of the field the sort is to be done on, i.e., an entry of 2 means sort on the second field. All information the user has entered will then be displayed and a request for confirmation will be made, as above.

After GSOSRT has completed the sort, the user is asked whether the file should be stored in ascending or descending order. After the user specifies his choice, the entire file is repacked in the requested order. When done, the statement "OPERATIONS COMPLETE" appears and the program will terminate.

This program can be run with the system booted under any version of 65D.

E. BASIC Disassembler

The BASIC Disassembler is a machine code disassembler written in BASIC. The program breaks down the machine code at specified intervals to an assembler format. It will print out the addresses in octal and hexadecimal, the machine code that was found at these addresses, the 65Ø2 assembler op-code, the operand which corresponds to the op-code, and special notes (for JMPs and JSRs). The program does not provide labels for branches, jumps, or other op-codes which could use labels.

The program is started by running the program DISASM. Then enter the addresses in decimal for the beginning and end of the disassembly. After the disassembly has been completed, "END OF DISASSEMBLY" will be displayed followed by the "OK" prompt of BASIC. This program can be run with the system booted under any version of 65D.

F. Data File Copier

The DISK!"LO___" and DISK!"PU___" commands provide the capability for transferring a program from one file to another. The utility program sorted in the file DATRAN provides a convenient method for copying data between two files.

To use this utility, type RUN"DATRAN and then enter an input file name, the device it is located on, an output file name, and the device it is located on (A, B, C, D). The input file is then copied onto the output file. Should the length (in tracks) of the input file exceed the length of the output file, the program will issue a warning and the user may at that point terminate the program or choose to continue and fill the output file as far as possible. This program assumes standard format data files, i.e., 8-page tracks on mini-floppies and 12-page tracks on 8" floppies.

This program may be run only with the system booted under version 3.2 of OS-65D (use with Tutorial Disk Two).

G. Program Listings

The last section of this chapter contains source listings for the six program files in the Extended 65D Utilities Package.

(i) RESEQUENCER (File: RSEQ)

```
10 REM * RSEQ V3.0 *
20 REM
30 REM (C) 1980 by OSI
40 REM ALL RIGHTS RESERVED
50 REM R. Whitesel 6/81
60 REM
70 FORX=709TO712:NAME$=NAME$+CHR$(PEEK(X)AND127):NEXT
80 IFNAME$="RSEQ"ORNAME$="NULL"THEN110
90 PRINTCHR$(7):PRINT"The ";NAME$;" Command is enabled,";
100 PRINT" Please disable it.":NEW
110 CC=PEEK(2073):POKE2073,096
120 FORX=1TO33:PRINT:NEXT
130 PP=80:IFPEEK(8993)=2THENPP=64
140 PP=(PP-20)/2
150 PRINTTAB(PP);"--------------
160 PRINTTAB(PP);": RSEQ V3.0 :
170 PRINTTAB(PP);"--------------
180 PRINT:PRINT:PRINT:PRINT
190 PRINT:PRINT"Enable or Disable RSEQ
200 INPUT"-          -              ";QA$
210 IFLEFT$(QA$,1)="E"THEN240
220 IFLEFT$(QA$,1)="D"THEN470
230 PRINT:PRINT"Do not understand ?":GOTO190
240 GOSUB630
250 BASE=(08*16+00)+(03*16+02)*256
260 IFSIZE=49152THENOFFSET=2560:DEST=48000:GOTO290
270 IFSIZE>=32768THENOFFSET=1280:DEST=31616:GOTO290
280 IFSIZE>=24576THENOFFSET=0000:DEST=23424
290 POKE132,(DEST-2)-INT((DEST-2)/256)*256
300 POKE133,INT((DEST-2)/256)
310 PRINT:PRINT"Working ";
320 FORX=0TO1151
330 T=PEEK(BASE+OFFSET+X)
340 POKEDEST+X,T
350 PRINT".";:IFPOS(X)>=60THENPRINTCHR$(13);
360 NEXT
370 AL=(DEST+28-1)-INT((DEST+28-1)/256)*256
380 AH=INT(DEST+28-1)/256
390 POKE546,AL:POKE547,AH
400 POKE709,ASC("R"):POKE710,ASC("S")
410 POKE711,ASC("E"):POKE712,ASC("Q")+128
420 FORX=1TO33:PRINT:NEXT
430 PRINT"RSEQ command now enabled.":PRINT
440 PRINT"System memory size found to be";SIZE;"bytes.":PRINT
450 POKE2073,CC
460 NEW
470 NA$="":FORX=709TO712:NA$=NA$+CHR$(PEEK(X)AND127):NEXT
480 IFNAME$="RSEQ"THEN510
490 PRINTCHR$(7):PRINT"RSEQ not enabled,";
500 PRINT" therfore cannot disable it.":POKE2073,CC:NEW
510 POKE709,ASC("N"):POKE710,ASC("U")
520 POKE711,ASC("L"):POKE712,ASC("L")+128
530 POKE546,108:POKE547,008
540 GOSUB630
```

```
550 POKE132,(SIZE-1)-INT((SIZE-1)/256)*256
560 POKE133,INT((SIZE-1)/256)
570 POKE8960,INT((SIZE-1)/256)-1:X=FRE(X)
580 FORX=1TO33:PRINT:NEXT
590 PRINT"RSEQ command now disabled.":PRINT
600 PRINT"System memory size found to be";SIZE;"bytes.":PRINT
610 POKE2073,CC
620 NEW
630 U1SR=PEEK(574):U2SR=PEEK(575)
640 POKE574,128:POKE575,46:DISK!"CA 2E79=39,2"
650 SIZE=USR(SIZE):IFSIZE<0THENSIZE=SIZE+65536
660 POKE574,U1SR:POKE575,U2SR
670 IFSIZE>(15*16+14)+((05*16+15)*256)THENRETURN
680 PRINTCHR$(7):PRINT"Not enough memory, only";SIZE;
690 PRINT"bytes found.":POKE2073,CC:NEW
```

(ii) REPACKER (File: REPACK)

```
10 DEF FNA(X)=10*INT(X/16)+X-16*INT(X/16)
20 PRINT"** BASIC PROGRAM REPACK UTILITY **"
21 PRINT
22 PRINT"** WARNING --- USE DRIVE A ONLY **"
30 PRINT:PRINT:PRINT
31 DISK!"SE A
40 INPUT"FILENAME";A$:GOSUB110
45 GOSUB300
50 IFA=0THENPRINT:PRINT"FILE NOT FOUND":GOTO100
60 POKE17022,A
70 DISK!"GO 4280
80 FR=PEEK(17022)+256*PEEK(17023)
90 PRINTFR"BYTES RECOVERED."
100 END
110 A=0:DR=11897
120 A$=A$+"       ":A$=LEFT$(A$,6)
130 DISK!"CA 2E79=12,1"
140 GOSUB160:IFA>0THENRETURN
150 DISK!"CA 2E79=12,2
160 FORI=1TO32:N$="":FORJ=0TO5
170 N$=N$+CHR$(PEEK(DR+(I-1)*8+J))
180 NEXTJ
190 IFN$<>A$THENNEXTI:RETURN
200 A=PEEK(DR+(I-1)*8+6):A=FNA(A):RETURN
300 PRINT:PRINT"REPACK FILE BY REMOVING:"
310 PRINT" 1> BLANK SPACES"
320 PRINT" 2> REM STATEMENTS"
330 PRINT" 3> BOTH BLANK SPACES AND REMS"
340 INPUT"ENTER YOUR CHOICE";B
350 IFB<1ORB>3GOTO300
360 POKE17023,B:RETURN
```

(iii) BUFFER CREATOR (File: BUFFER)

```
100 REM                  ::::::::::
110 REM                  : BYTEV1 :
120 REM                  ::::::::::
130 REM
140 REM
150 REM      (C) 1980 BY OHIO SCIENTIFIC, INC.
160 REM      ALL RIGHTS RESERVED / WRITTEN BY
170 REM       PAUL A. JOVIAK   8/80   AURORA
180 REM
190 REM
200 GOSUB 1410
210 POKE 2073,96 : REM CTRL-C OFF
220 FOR X=1 TO 33: PRINT: NEXT
230 PRINT "::::::::::::::::::::::::::"
240 PRINT ": OS-65D BUFFER CREATOR :"
250 PRINT "::::::::::::::::::::::::::": FOR X=1 TO 5: PRINT: NEXT
260 PRINT: PRINT "Enable or Disable BYTE command"
270 INPUT          "-          -                          ";QA$: PRINT
280 IF LEFT$(QA$,1)="E" GOTO 350
290 IF LEFT$(QA$,1)="D" GOTO 1150
300 PRINT: PRINT "WHAT ?": PRINT: GOTO 230
310 REM
320 REM
330 REM FIND OUT HOW MUCH MEMORY
340 REM
350 MEMSIZ=PEEK(8960)
360 REM
370 REM SET UP FOR TRANSFER
380 REM
390 IF MEMSIZ >=95 GOTO 410
400 PRINT: PRINT "Not enough memory, 24K RAM min.": NEW
410 IF MEMSIZ=95 GOTO 470
420 IF MEMSIZ <=127 THEN MEMSIZ=127: GOTO 470
430 MEMSIZ=191
440 REM
450 REM POKE SYSTEM MEMORY SIZE
460 REM
470 AD=((MEMSIZ-3)*256)-10: REM DECIMAL OF MEM TOP
480 REM
490 POKE 132,AD-INT(AD/256)*256: POKE 133,INT(AD/256)
500 T=FRE(X): REM FORCE NEW MEM ADDR
510 REM
520 REM SET BASE = START OF IMBEDDED CODE
530 REM
540 IF MEMSIZ=95 THEN BASE=12688
550 IF MEMSIZ=127 THEN BASE=13712
560 IF MEMSIZ=191 THEN BASE=14736
565 BASE=BASE+256
```

```
570 REM
580 REM MOVE BYTE CODE INTO PLACE
590 REM
600 PRINT: PRINT "Working ";
610 FOR X=0 TO 1032: T=PEEK(X+BASE): POKE AD+1+X,T
620 PRINT ".";: IF PEEK(22) > 60 THEN PRINT CHR$(13);
630 NEXT: PRINT CHR$(13);SPC(61);CHR$(13);
640 REM
650 REM DETERMIN DISPATCH ADDR.
660 REM
670 IF MEMSIZ=095 THEN DH=092
680 IF MEMSIZ=127 THEN DH=124
690 IF MEMSIZ=191 THEN DH=188
700 REM
710 REM POKE DISPATCH ADDR INTO PLACE
720 REM
730 POKE 546,07: POKE 547,DH
740 REM
750 REM POKE RESERVED WORD "BYTE" INTO PLACE
760 REM
770 POKE 709,ASC("B"): POKE 710,ASC("Y")
780 POKE 711,ASC("T"): POKE 712,ASC("E")+128
790 REM
800 POKE2073,173 : REM CTRL-C ON
1090 REM
1100 PRINT "BYTE command now enabled.": NEW
1110 REM
1120 REM
1130 REM GOTO MACHINE CODE TO SET MEMSIZ
1140 REM
1150 POKE 574,160: POKE 575,062
1160 X=USR(X): CLEAR
1170 REM
1180 REM POKE "NULL" BACK
1190 REM
1200 POKE 709,ASC("N"): POKE 710,ASC("U")
1210 POKE 711,ASC("L"): POKE 712,ASC("L")+128
1220 REM
1230 REM POKE NULL DISPATCH BACK
1240 REM
1250 POKE 546,108: POKE 547,008
1260 REM
1380 REM
1390 POKE 2073,173 : REM CTRL-C ON
1400 PRINT: PRINT "BYTE command now disabled.": NEW
1410 PRINT:QA$=CHR$(PEEK(709))
1420 IF QA$="R" THEN PRINT"RSEQ is enabled, can not enable BYTE":NEW
1430 IF QA$="E" THEN PRINT"EDIT is enabled, can not enable BYTE":NEW
1440 RETURN
```

(iv) GENERALIZED STRING ORIENTED SORT (File: GSOSRT)

```
10 REM OS65D V3.1 GENERALIZED SORT UTILITY
15 REM            9/21/79
20 REM
25 FORI=1TO5:PRINT:NEXT:BY=2048
30 DEF FNA(X)=NB*X+1
35 PRINT"** OS-65D V3.2 GENERALIZED SORT UTILITY **"
40 PRINT:PRINT
45 INPUT"ENTER DATA FILE NAME";A$:T=LEN(A$):IFT>0THENT=ASC(A$)
50 IFT<65ORT>90GOTO45
55 FI$=A$:PRINT:PRINT
60 INPUT"IS THIS FILE A MDMS MASTER FILE";M$:PRINT
65 M$=LEFT$(M$,1):IFM$<>"Y"GOTO110
70 FI$=FI$+"      ":FI$=LEFT$(FI$,5)+"0"
75 DISK OPEN,6,FI$:POKE12076,6:POKE12042,BY/64
80 INPUT#6,A$,NB,NF,PH,EN:IFEN=1GOTO49999
85 EN=EN-1:DISK GET,1
90 PRINT:PRINT"SORT ON WHICH OF THE FOLLOWING FIELDS:"
95 FORI=1TONF:INPUT#6,A$,B$:PRINTI">  ";A$:NEXT
100 INPUT"WHICH ONE";SE:IFSE<1ORSE>NFGOTO100
105 PRINT:GOTO175
110 INPUT"IS THIS FILE 'RANDOM' OR 'SEQUENTIAL'";A$
115 DISK OPEN,6,FI$
120 R$="R":IFLEFT$(A$,1)="S"THENR$="S":GOTO145
125 INPUT"NUMBER OF BYTES PER RECORD";A
130 IFA<1ORA>255GOTO125
135 B=BY/A:IFB<0ORB>255GOTO125
140 POKE12076,6:POKE12042,B
145 INPUT"NUMBER OF FIELDS PER RECORD";NF
150 IFNF<1ORNF>100GOTO145
155 IFNF=1THENSE=1:GOTO170
160 INPUT"SORT ON WHICH FIELD";SE
165 IFSE<1ORSE>NFGOTO160
170 INPUT"HOW MANY RECORDS TO SORT";EN:IFEN<1GOTO170
175 PRINT:PRINT:PRINT"DESCRIPTION CONFIRMATION:":PRINT
180 A$="NON-MDMS":IFM$="Y"THENA$="MDMS MASTER"
185 PRINT"FILE TYPE"TAB(18)":  ";A$
190 IFM$="Y"GOTO210
195 A$="RANDOM":IFR$="S"THENA$="SEQUENTIAL"
200 PRINT"DATA TYPE"TAB(18)":  "A$
205 IFR$<>"S"THENPRINT"BYTES PER RECORD"TAB(18)": ";A
210 PRINT"NUMBER OF FIELDS"TAB(18)": "NF
215 PRINT"SORT ON FIELD"TAB(18)": "SE
220 PRINT"RECORDS TO SORT"TAB(18)": "EN
225 PRINT:INPUT"IS THIS CORRECT";Y$
230 IFLEFT$(Y$,1)="N"THENRUN
235 DIMR(EN),L$(EN),S$(EN,NF)
240 PRINT:PRINT:PRINT"SORTING---"
245 FORI=1TOEN:IFM$="Y"THEN DISK GET,FNA(I):GOTO255
250 IFR$="R"THEN DISK GET,I-1
255 FORJ=1TONF:INPUT#6,S$(I,J):NEXTJ
260 L$(I)=S$(I,SE):R(I)=I:NEXTI
265 T=0:FORI=1TOEN:IFVAL(L$(I))<>0THENT=T+1
270 NEXT:A$="":B$=" ":IFT<EN/3GOTO285
```

```
275 FORI=1TOEN:L$(I)=STR$(INT(VAL(L$(I))*1000)):NEXT
280 A$=" ":B$=""
285 SL=0:FORI=1TOEN:A=LEN(L$(I)):IFA>SLTHENSL=A
290 NEXT:FORI=1TOEN:C$=L$(I)
295 IFLEN(C$)<SLTHENC$=A$+C$+B$:GOTO295
300 L$(I)=C$:NEXT
305 FORI=2TOEN:IFL$(I-1)<=L$(I)GOTO330
310 A$=L$(I):J=I:RO=R(I)
315 J=J-1:IFJ<1ORL$(J)<=A$GOTO325
320 L$(J+1)=L$(J):R(J+1)=R(J):GOTO315
325 J=J+1:L$(J)=A$:R(J)=RO
330 NEXT:PRINT:PRINT:PRINT"SORT COMPLETE.  WOULD YOU LIKE THE"
335 PRINT"DATA TO BE STORED IN 'ASCENDING' OR"
340 INPUT"'DESCENDING' ORDER";Y$:A=1:B=EN:C=1
345 IFLEFT$(Y$,1)="D"THENA=EN:B=1:C=-1
350 IFR$="S"THENDISKOPEN,6,FI$
355 FORI=ATOBSTEPC:IFM$="Y"THEN DISK GET,FNA(I):GOTO365
360 IFR$="R"THENDISK GET,I-1
365 FORJ=1TONF:PRINT#6,S$(R(I),J):NEXTJ
370 IFR$="R"ORM$="Y"THEN DISK PUT
375 NEXTI:IFR$="S"THEN DISK PUT
49999 PRINT:PRINT"OPERATIONS COMPLETE.":END
```

(v) BASIC DISASSEMBLER (File: DISASM)

```
10 INPUT"INPUT STARTING DISASSEMBLY ADDRESS IN DECIMAL";A1
20 INPUT"INPUT ENDING ADDRESS";A2
30 PRINTTAB(10);"OCTAL";TAB(18);"HEX";
35 PRINTTAB(23);"CODE";TAB(33);"OP";TAB(40);
40 PRINT"OPRAND":PRINT
60 DIMOT$(151),MN$(56),N$(15)
80 GOSUB1160:REM INITIALIZE ARRAYS
90 W=6:BA%=8:OC%=A1:GOSUB1530
100 OA$=OC$
110 W=4:BA%=16:OC%=A1:GOSUB1530
120 AD$=OC$
130 W=2:OC%=PEEK(A1):GOSUB1530
140 GOSUB1060
150 IFFL=1THEN200
160 GOSUB230
170 A1=A1+VAL(NB$)
180 IFA1<=A2THEN90
190 GOTO220
200 PRINTTAB(10);OA$;TAB(18);AD$;TAB(23);OC$
210 A1=A1+1:GOTO90
220 PRINT"END OF DISASSEMBLY":END
230 W=2
240 IFTY$<>"IM"THEN290
250 OC%=PEEK(A1+1):GOSUB1530
260 PRINTTAB(10);OA$;TAB(18);AD$;
265 PRINTTAB(23);O1$;OC$;TAB(33);MN$(VAL(PT$));
270 PRINTTAB(40);"#$";OC$
280 RETURN
290 IFTY$<>"Z "THEN340
300 OC%=PEEK(A1+1):GOSUB1530
310 PRINTTAB(10);OA$;TAB(18);AD$;
315 PRINTTAB(23);O1$;OC$;TAB(33);MN$(VAL(PT$));
320 PRINTTAB(40);"$";OC$
330 RETURN
340 IFTY$<>"ZX"THEN390
350 OC%=PEEK(A1+1):GOSUB1530
360 PRINTTAB(10);OA$;TAB(18);AD$;
365 PRINTTAB(23);O1$;OC$;TAB(33);MN$(VAL(PT$));
370 PRINTTAB(40);"$";OC$;",X"
380 RETURN
390 IFTY$<>"ZY"THEN440
400 OC%=PEEK(A1+1):GOSUB1530
410 PRINTTAB(10);OA$;TAB(18);AD$;
415 PRINTTAB(23);O1$;OC$;TAB(33);MN$(VAL(PT$));
420 PRINTTAB(40);"$";OC$;",Y"
430 RETURN
440 IFTY$<>"A "THEN530
450 OC%=PEEK(A1+1):GOSUB1530
460 PRINTTAB(10);OA$;TAB(18);AD$;TAB(23);O1$;OC$;
470 CO$=OC$:OC%=PEEK(A1+2):GOSUB1530
480 PRINTOC$;TAB(33);MN$(VAL(PT$));TAB(40);"$";OC$;CO$;
490 IFO1$="4C"ORO1$="6C"THENPRINTTAB(55);"** JMP **":GOTO520
500 IFO1$="20"THENPRINTTAB(55);"** JSR **":GOTO520
510 PRINT
520 RETURN
```

```
530 IFTY$<>"AX"THEN590
540 OC%=PEEK(A1+1):GOSUB1530
550 PRINTTAB(10);OA$;TAB(18);AD$;TAB(23);O1$;OC$;
560 CO$=OC$:OC%=PEEK(A1+2):GOSUB1530
570 PRINTOC$;TAB(33);MN$(VAL(PT$));TAB(40);"$";OC$;CO$;",X"
580 RETURN
590 IFTY$<>"AY"THEN650
600 OC%=PEEK(A1+1):GOSUB1530
610 PRINTTAB(10);OA$;TAB(18);AD$;TAB(23);O1$;OC$;
620 CO$=OC$:OC%=PEEK(A1+2):GOSUB1530
630 PRINTOC$;TAB(33);MN$(VAL(PT$));TAB(40);"$";OC$;CO$;",Y"
640 RETURN
650 IFTY$<>"IX"THEN700
660 OC%=PEEK(A1+1):GOSUB1530
670 PRINTTAB(10);OA$;TAB(18);AD$;
675 PRINTTAB(23);O1$;OC$;TAB(33);MN$(VAL(PT$));
680 PRINTTAB(40);"($";OC$;",X)"
690 RETURN
700 IFTY$<>"IY"THEN750
710 OC%=PEEK(A1+1):GOSUB1530
720 PRINTTAB(10);OA$;TAB(18);AD$;
725 PRINTTAB(23);O1$;OC$;TAB(33);MN$(VAL(PT$));
730 PRINTTAB(40);"($";OC$;"),Y"
740 RETURN
750 IFTY$<>"IP"THEN780
760 PRINTTAB(10);OA$;TAB(18);AD$;
765 PRINTTAB(23);O1$;TAB(33);MN$(VAL(PT$))
770 RETURN
780 IFTY$<>"I "THEN860
790 OC%=PEEK(A1+1):GOSUB1530
800 PRINTTAB(10);OA$;TAB(18);AD$;TAB(23);O1$;OC$;
810 CO$=OC$:OC%=PEEK(A1+2):GOSUB1530
820 PRINTOC$;TAB(33);MN$(VAL(PT$));TAB(40);"($";OC$;CO$;")";
830 IFO1$="6C"THENPRINTTAB(55);"** JMP **":GOTO850
840 PRINT
850 RETURN
860 IFTY$<>"R "THEN1030
870 OC%=PEEK(A1+1):GOSUB1530
880 SS=PEEK(A1+1)
890 IF SS<128 THEN OC%=A1+2+SS
900 IF SS>=128 THEN OC%=A1+SS-254
960 PRINTTAB(10);OA$;TAB(18);AD$;
965 PRINTTAB(23);O1$;OC$;TAB(33);MN$(VAL(PT$));
975 W=4
980 GOSUB1530
990 PRINTTAB(40);"$";OC$
1020 RETURN
1030 PRINTTAB(10);OA$;TAB(18);AD$;
1035 PRINTTAB(23);O1$;TAB(33);MN$(VAL(PT$));TAB(40);
1040 PRINT"A"
1050 RETURN
1060 FL=0:J0=1:J1=151
1070 J=INT((J0+J1)/2)
1080 OZ$=MID$(OT$(J),1,2)
1082 IF OC$=OZ$THEN 1090
```

```
1083 IF ABS(J1-JO)<2THENFL=1:RETURN
1084 IF OC$<OZ$THEN J1=J
1086 IF OC$>OZ$THEN JO=J
1089 GOTO 1070
1090 O1$=OC$
1100 TY$=MID$(OT$(J),3,2)
1110 NB$=MID$(OT$(J),5,1)
1120 PT$=MID$(OT$(J),6,2)
1130 RETURN
1160 REM TABLE DATA
1170 FORI=1TO151:READOT$(I):NEXTI
1180 FORI=1TO56:READMN$(I):NEXTI
1190 FORI=0TO15:READN$(I):NEXTI
1195 RETURN
1200 DATA"00IP111","01IX235","05Z 235","06Z 203","08IP137","09IM235"
1210 DATA"0AAC103","0DA 335","0EA 303","10R 210","11IY235","15ZX235"
1220 DATA"16ZX203","18IP114","19AY335","1DAX335","1EAX303","20A 329"
1230 DATA"21IY202","24Z 207","25Z 202","26Z 240","28IP139","29IM202"
1240 DATA"2AAC140","2CA 307","2DA 302","2EA 340","30R 208","31IY202"
1250 DATA"35ZX202","36ZX240","38IP145","39AY302","3DAX302","3EAX340"
1260 DATA"40IP142","41IX224","45Z 224","46Z 233","48IP136","49IM224"
1270 DATA"4AAC133","4CA 328","4DA 324","4EA 333","50R 212","51IY224"
1280 DATA"55ZX224","56ZX233","58IP116","59AY324","5DAX324","5EAX333"
1290 DATA"60IP143","61IX201","65Z 201","66Z 241","68IP138","69IM201"
1300 DATA"6AAC141","6CI 328","6DA 301","6EA 341","70R 213","71IY201"
1310 DATA"75ZX201","76ZX241","78IP147","79AY301","7DAX301","7EAX341"
1320 DATA"81IX248","84Z 250","85Z 248","86Z 249","88IP123","8AIP155"
1330 DATA"8CA 350","8DA 348","8EA 349","90R 204","91IY248","94ZX250"
1340 DATA"95ZX248","96ZY249","98IP153","99AY348","9AIP156","9DAX348"
1350 DATA"A0IM232","A1IX230","A2IM231","A4Z 232","A5Z 230","A6Z 231"
1360 DATA"A8IP152","A9IM230","AAIP151","ACA 332","ADA 330","AEA 331"
1370 DATA"B0R 205","B1IY230","B4ZX232","B5ZX230","B6ZY231","B8IP117"
1380 DATA"B9AY330","BAIP154","BCAX332","BDAX330","BEAY331","C0IM220"
1390 DATA"C1IX218","C4Z 220","C5Z 218","C6Z 221","C8IP127","C9IM218"
1400 DATA"CAIP122","CCA 320","CDA 318","CEA 321","D0R 209","D1IY218"
1410 DATA"D5ZX218","D6ZX221","D8IP115","D9AY218","DDAX318","DEAX321"
1420 DATA"E0IM219","E1IX244","E4Z 219","E5Z 244","E6Z 225","E8IP126"
1430 DATA"E9IM244","EAIP134","ECA 319","EDA 344","EEA 325","F0R 206"
1440 DATA"F1IY244","F5ZX244","F6ZX225","F8IP146","F9AY344","FDAX344"
1450 DATA"FEAX325"
1460 DATA"ADC","AND","ASL","BCC","BCS","BEQ","BIT","BMI","BNE","BPL"
1470 DATA"BRK","BVC","BVS","CLC","CLD","CLI","CLV","CMP","CPX","CPY"
1480 DATA"DEC","DEX","DEY","EOR","INC","INX","INY","JMP","JSR","LDA"
1490 DATA"LDX","LDY","LSR","NOP","ORA","PHA","PHP","PLA","PLP","ROL"
1500 DATA"ROR","RTI","RTS","SBC","SEC","SED","SEI","STA","STX","STY"
1510 DATA"TAX","TAY","TYA","TSX","TXA","TXS"
1520 DATA "0","1","2","3","4","5","6","7","8","9"
1525 DATA "A","B","C","D","E","F"
1530 REM DECIMAL TO HEX
1560 OC$=""
1570 Q=INT(OC%/BA%)
1580 R=OC%-BA%*Q
1590 OC$=N$(R)+OC$:OC%=Q
1600 IF Q<>0 THEN 1570
1610 IFLEN(OC$)<W THEN OC$="0"+OC$:GOTO1610
1620 RETURN
```

(vi) DATA FILE COPIER (File: DATRAN

```
10 REM    GENERALIZED FILE TRANSFER UTILITY
20 REM                  10/14/79
30 P$="/8":S$="327E":DR$="12"
40 FORI=1TO5:PRINT:NEXT
60 DEF FNB(X)=10*INT(X/16)+X-16*INT(X/16)
70 PRINT"** DATA FILE TRANSFER UTILITY **":PRINT:PRINT
80 INPUT"INPUT FILE NAME";IN$:T=LEN(IN$):IFT>0THENT=ASC(IN$)
90 IFT<65ORT>90GOTO80
100 GOSUB1000:D1$=D$:PRINT:PRINT
110 DISK!"SE "+D$:N1$=IN$:GOSUB2000
120 IFS=0THENPRINT:PRINT"FILE NOT FOUND.":GOTO49999
130 IB=T0:IE=T9
140 PRINT:INPUT"OUTPUT FILE NAME";OU$:T=LEN(OU$):IFT>0THENT=ASC(OU$)
150 IFT<65ORT>90GOTO140
160 GOSUB1000:PRINT:PRINT
170 IFD$<>D1$THENDISK!"SE "+D$
180 N1$=OU$:GOSUB2000:IFS=0GOTO120
190 IF (IE-IB) > (T9-T0) THENGOSUB3000
200 OF=IB-T0:PRINT:INPUT"TYPE 'Y' WHEN READY";Y$
210 FORI=IBTOIE
220 IFD$<>D1$THENDISK!"SE "+D1$
230 T=I:GOSUB4000
240 DISK!"CA "+S$+"="+T$+",1"
250 IFD$<>D1$THENDISK!"SE "+D$
260 T=I-OF:GOSUB4000
270 DISK!"IN "+T$
280 DISK!"SA "+T$+",1="+S$+P$
290 NEXTI
300 PRINT:PRINT"OPERATIONS COMPLETE.":GOTO49999
1000 PRINT:INPUT"LOCATED ON WHICH DEVICE (A,B,C,D)";D$
1010 D=LEN(D$):IFD>0THEND=ASC(D$)
1020 IFD<65ORD>68GOTO1000
1030 D$=CHR$(D):RETURN
2000 N1$=N1$+"       ":N1$=LEFT$(N1$,6)
2010 Z$=DR$+",1":GOSUB2040:IFS=1THENRETURN
2020 Z$=DR$+",2":GOTO 2040
2040 DISK!"CA 2E79="+Z$
2050 S=0:FORI=11897TO12145STEP8:N$=""
2060 FORJ=0TO5:N$=N$+CHR$(PEEK(I+J)):NEXT
2070 IFN$<>N1$THENNEXTI:RETURN
2080 S=1:T0=PEEK(I+6):T9=PEEK(I+7)
2090 T0=FNB(T0):T9=FNB(T9):RETURN
3000 PRINT:PRINT:PRINT"   <<< WARNING >>>":PRINT
3010 PRINT"INPUT FILE LENGTH EXCEEDS OUTPUT FILE LENGTH"
3020 PRINT"BY"ABS(T9-T0-IE+IB)"TRACKS.  PROGRAM MAY ONLY"
3030 PRINT"OUTPUT FILE TO"T9-T0+1"TRACKS."
3040 PRINT:INPUT"CONTINUE (Y OR N)";Y$:IFY$="N"THENEND
3050 IE=IB+T9-T0:RETURN
4000 T$=MID$(STR$(T),2):IFT<10THENT$="0"+T$
4010 RETURN
49999 PRINT:PRINT:END
```

Appendix 1

The Utility Programs

(Detailed Information and Listing)

The following pages contain detailed information on all of the programs found on the OS-65D V3.3 System Disk (Tutorial Disk 5). For each program, the information will normally be in three parts. Part one is a description of the program and its use. Part two is a fact sheet with the name, size, location and other notes concerning the program. (A sample fact sheet is on the next page.) Finally, part three is a listing (when appropriate) of the program as it appears on the mini-floppy version of this disk.

It has already been mentioned that four of these programs (DIR, CREATE, RENAME and DELETE) are redundant in the sense that they perform functions that are also performed by the BEXEC* program. These four programs are included on this disk primarily for the convenience of experienced programmers who may have used them with earlier versions of OS-65D.

SAMPLE FACT SHEET

File name : As it appears in the directory

File type : Assembler source/BASIC program/Object code

	Mini-floppy specifications	8" floppy specifications
Location:	The track location of the file	
Length :	The length in tracks of the files	
Buffers for:	Is there a buffer for	
Device #6:		YES/NO
Device #7:		YES/NO
Other :	Number of bytes/pages other than a device #6 or #7 buffer	

Mini-floppy to 8" floppy program conversions

The lines in the BASIC program listings which can be changed to make the mini-floppy listing agree with the programs on the 8" floppies.

OS65D3

The entry "OS65D3" in the directory is not an executable program. It is the object code for the systems on disk (Assembler, BASIC, ...). These nine tracks, numbered from zero through eight (fourteen tracks on mini-floppy, numbered from zero through thirteen) and the entry, "BEXEC*", are the most important pieces of software on a disk. Do not delete these entries from the directory and never initialize the tracks where these entries reside.

The next two pages give a complete breakdown of the object code located on these tracks.

Program	Track	Sector or Format	Start of Transfer	Length in Pages	Go Address	Comments
OS-65D V3.3 - Part 1	Ø	1	22ØØ	8	22ØØ	
OS-65D V3.3 - Part 2	1	1	2AØØ	8		
BASIC - Part 1	2	1	Ø2ØØ	8		
BASIC - Part 2	3	1	ØAØØ	8		
BASIC - Part 3	4	1	12ØØ	8		
BASIC - Part 4	5	1	1AØØ	8		
BASIC - Part 5	6	1	22ØØ	1		
OS-65D V3.3 - Part 3	6	2	32ØØ	1		
OS-65D V3.3 - Zero Page	6	3	ØØØØ	1		
TRACK Ø UTILITY	6	4	Ø2ØØ	5	Ø2ØØ	
ASSEMBLER - Part 1	7	1	Ø2ØØ	8		
ASSEMBLER - Part 2	8	1	ØAØØ	8		
ASSEMBLER/EXTENDED MONITOR	9	1	12ØØ	8		
EXTENDED MONITOR	1Ø	1	1AØØ	8		
EXTENDED MONITOR	11	1	22ØØ	1		Rest of Track 11 used for storage
DIRECTORY - Page 1	12	1	2E79	1		
DIRECTORY - Page 2	12	2	2E79	1		
BASIC OVERLAYS	12	3	2ØC4	1		
PUT/GET OVERLAYS	12	4	2E79	1		
OS-65D V3.3 - Part 4	13	1	3274	8		
BEXEC*	14	1	3A79	8		
COMPAR/TRACKØ	39	1	Ø2ØØ	5	Ø2ØØ	
UTILITY	39	2	2ØØØ	2		

8" Disk Directory

Program	Track	Sector or Format	Start of Transfer	Length in Pages	Go Address	Comments
OS-65D V3.3 - Part 1	Ø	1	22ØØ	8	22ØØ	
OS-65D V3.3 - Part 2	1	1	2AØØ	5		
TRACK Ø UTILITY	1	2	Ø2ØØ	5	Ø2ØØ	
OS-65D V3.3 - Part 3	1	3	318Ø	1		
OS-65D V3.3 - Zero Page	1	4	ØØØØ	1		
BASIC - Part 1	2	1	Ø2ØØ	B/11		
BASIC - Part 2	3	1	ØDØØ	B/11		
BASIC - Part 3	4	1	18ØØ	B/11		
ASM/EM - Part 1	5	1	Ø2ØØ	B/11		
ASM/EM - Part 2	6	1	ØDØØ	C/12		
ASM/EM - Part 3	7	1	19ØØ	7		Rest of Track 7 used for storage
DIRECTORY - Page 1	8	1	2E79	1		
DIRECTORY - Page 2	8	2	2E79	1		
BASIC OVERLAYS	8	3	2ØC4	1		
PUT/GET OVERLAYS	8	4	2E79	1		
OS-65D V3.3 - Part 4	8	5	3274	8		
BEXEC*	9	1	3A79	B/11		
COMPAR/TRACKØ UTILITY	33	1	Ø2ØØ	6	Ø2ØØ	

File name : OS65D3

File type : OBJECT CODE

	Mini-floppy specifications	8" floppy specifications
Location:	Tracks Ø-13	Tracks Ø-8
Length :	14 Tracks	9 Tracks
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

NOT APPLICABLE

BEXEC*

The OS-65D operating system is intelligent in the sense that it runs one program on boot up. This special program is called "BEXEC*" on most disks. (The equivalent program on Tutorial Disks three and four is called "MENU" for simplicity.) Generally, the first few lines of this BASIC program, BEXEC*, are used for system considerations such as memory configurations, I/O device setup, etc. The next section of the BEXEC* program displays a disk title, numbered options and then a request for your response such as, "Type the number of your selection and depress RETURN?". Typing in the number of your selection (generally followed by the RETURN key) will select the program corresponding to your choice. A series of checks for the legality of your entry along with the processing of your selection will take place at this time. If an illegal entry is made, BEXEC* will make a statement referring to the legality of your entry and redisplay the menu waiting for a legal response.

There are two ways to exit the BEXEC* program into the operating system (DOS). The first way is to use a password when you are asked for information. The password will normally be "UNLOCK" or "PASS" ("PASS" in the case of the Tutorial Disk 5). The second method is to select the menu selection which will open or exit to the local operating system. Many disks will not have a menu selection for exiting in which case UNLOCK or PASS must be used. For others the menu selection is the only way to open the system. After the exit function has been selected correctly the system will be unlocked, which means that all available functions will be enabled and the OK prompt of BASIC will appear on the screen along with a message stating that the system has been opened.

One can always return to the initial menu by typing RUN"BEXEC*" followed by RETURN. In the case of Tutorial Disks three and four, one must type RUN"MENU" followed by RETURN.

File name : BEXEC*

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Tracks 14-16	Tracks 9-12
Length :	3 Tracks	4 Tracks
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

```
    5 POKE 133,122:CLEAR:POKE14172,8:POKE14170,16
   20 X=PEEK(10950):POKE8993,X:POKE8994,X:DIMAL%(76)
 2056 INPUT"and depress RETURN (1 to 76)";T1
 2060 IF T1<1 OR T1>76 THEN 2050
 2076 INPUT"and depress RETURN (1 to 68)";N
 2080 IF N<1 OR N>68 THEN 2070
 2125 GOSUB50010:PG=12:P$="123456789ABCDEF"
 2130 INPUT"How many pages per track (1 to 12) <12>";PG
 2135 IF PG=0 THEN PG=12
 2140 IF PG<1 OR PG>12 THEN 2125
 2156 J$="3A7E":IFPEEK(8999)=49 THEN J="317E"
 5020 GOSUB10200:DISK!"SE A":DISK!"CA 7C00=07,2"
 5030 DISK!"CA 7D00=07,3":DISK!"CA 7E00=07,4":DISK!"CA 7F00=07,5"
 5033 IF S=6 THEN DISK!"CA 7C00=07,6"
 5035 GOSUB5100:DISK"GO 2768":DISK!"SA 08,1=7C00/1"
 5040 DISK!"SA 08,2=7D00/1":DISK!"SA 08,3=7E00/1"
 5050 DISK!"SA 08,4=7F00/1":IF S=6 THEN GOSUB6000
 6000 P$="12":T1=1:T2=5:GOSUB2153
 6010 T1=9:T2=28:GOTO2153
 7000 L=12670:IF PEEK(8999)=58 THEN L=14974
 7040 L=L+N*3072:N=N+1:GOSUB50010:ONNGOTO71000,72000,73000
20000 NF=0:E$="N":FORK=0 TO 76:AL%(K)=0:NEXT
20005 DISK!"CA 2E79=08,1":GOSUB20100:DISK!"SA 08,1=2E79/1"
20010 DISK!"CA 2E79=08,2":GOSUB20100:DISK!"SA 08,2=2E79/1"
```

```
1 REM:New BEXEC*  July 16,1981
5 POKE133,91:CLEAR:POKE14172,8:POKE14170,16
10 POKE2888,0:POKE8722,0
20 X=PEEK(10950):POKE 8993,X:POKE8994,X:DIMAL%(39)
30 IFPEEK(57088)=223THENPOKE9794,37
40 DEFFNA(X)=10*INT(X/16)+X-16*INT(X/16)
50 DEFFNB(X)=16*INT(X/10)+X-10*INT(X/10)
100 GOSUB50000
110 PRINT:PRINT"OS-65D  Tutorial disk five":PRINT:PRINT
120 PRINT" 1 > Directory"
130 PRINT" 2 > Create a new file"
140 PRINT" 3 > Change a file name"
150 PRINT" 4 > Delete file from diskette"
160 PRINT" 5 > Create blank data diskette"
170 PRINT" 6 > Create data diskette with files"
180 PRINT" 7 > Create buffer space for data files"
190 PRINT" 8 > Single or dual disk drive copier"
200 PRINT" 9 > Enter OS-65D system"
890 PRINT:PRINT
900 PRINT"Type the number of your selection ";
910 INPUT"and depress RETURN ";S$:IFS$="PASS"ORS$="9"THEN60000
915 IFLEN(S$)>1THENRUN
920 S=INT(VAL(S$)):IFS<1ORS>8THENRUN
980 GOSUB50010
989 PRINT"     ";
990 ONSGOSUB1000,2000,3000,4000,5000,5000,7000,8000
998 IFP$="PASS"THEN60000
999 GOTO100
1000 PRINT"Directory utility":PRINT:PRINT:PRINT"Directory of";
1005 GOSUB10000:GOSUB50010:GOSUB10100:GOSUB50010
1010 PRINT#DV,"   -- Directory --":PRINT#DV
1020 PRINT#DV,"File name   Track range"
1030 PRINT#DV,"----------------------":GOSUB20000
1040 PRINT#DV:PRINT#DV,NF;"Entries free out of 64"
1050 PRINT:GOTO10200
2000 PRINT"Create utility":PRINT:PRINT
2005 PRINT"Create a file on";:GOSUB10000
2010 GOSUB50010:PRINT"Type in the name of the file you ";
2015 PRINT"want to create and":PRINT
2016 INPUT"depress RETURN (1-6 Characters) ";A$:IFLEN(A$)>6THEN2010
2018 IFA$=""THENRUN
2020 IFLEN(A$)<6THENA$=A$+" ":GOTO2020
2030 IFMID$(A$,1,1)<"A"ORMID$(A$,1,1)>"Z"THEN2010
2040 S=5:GOSUB20000:IFE$="Y"THEN40100
2045 IFNF=0THEN40200
2070 GOSUB50010:PRINT"Type in the number of ";
2075 PRINT"tracks in this file":PRINT
2076 INPUT"and depress RETURN (1 to 27) ";N
2080 IFN<1ORN>27THEN2070
2090 FORT1=0TO39-N+1:FORTS=0TON-1
2095 IFAL%(T1+TS)THEN2110
2100 NEXTTS
2102 PRINT:PRINT"** TRACKS AVAILABLE FOR ";A$;" **"
2103 T2=T1+N-1:N=0
2104 PRINT:PRINT:GOTO2120
```

```
2110 NEXT T1
2112 PRINT:PRINT"** NO ROOM FOR ";A$;" **"
2114 PRINT:GOTO10200
2120 PRINT"Do you want to initialize ":PRINT
2121 INPUT"these tracks (Yes or No) <Yes> ";B$
2123 IFLEFT$(B$,1)="N"THEN2200
2125 GOSUB50010:PG=8:P$="123456789ABCDEF"
2130 INPUT"How many pages per track (1 to 8) <8> ";PG
2135 IFPG=0THENPG=8
2140 IFPG<1ORPG>8THEN2125
2150 P$=MID$(P$,PG,1)
2152 PRINT"Initializing:"
2153 FORI=T1TOT2:T$=RIGHT$(STR$(I+100),2)
2155 PRINT"      Track ";T$
2160 DISK!"IN "+T$
2162 POKE10304,169:POKE10305,32:POKE10549,201:POKE10550,32
2164 DISK!"SA "+T$+",1=D000/"+P$
2166 POKE 10304,177:POKE10305,254:POKE10549,209:POKE10550,254
2167 NEXT
2170 IFS=6THENRETURN
2200 S=2:GOTO20000
3000 PRINT"Rename utility":PRINT:PRINT
3005 PRINT"Rename a file on";:GOSUB10000
3010 GOSUB50000:PRINT"Type in the name of the file that you";
3015 PRINT" want to rename":PRINT
3016 INPUT"and depress RETURN (1-6 Characters) ";A$
3017 IFLEN(A$)>6THEN3010
3018 IFA$=""THENRUN
3020 IFLEN(A$)<6THENA$=A$+" ":GOTO3020
3030 S=5:GOSUB20000:IFE$="N"THEN40000
3035 O$=A$:PRINT
3040 GOSUB50000:PRINT"Type in the name that will replace ";
3044 PRINTCHR$(34);O$;CHR$(34)" in the directory":PRINT
3045 INPUT"and depress RETURN (1-6 Characters) ";A$
3047 IFLEN(A$)>6THEN3040
3050 IFLEN(A$)<6THENA$=A$+" ":GOTO3050
3055 IFLEFT$(A$,1)<"A"ORLEFT$(A$,1)>"Z"THEN3040
3056 IFO$=A$THEN40100
3060 GOSUB20000:IFE$="Y"THEN40100
3070 S=3:GOTO20000
4000 PRINT"Delete utility":PRINT:PRINT
4005 PRINT"Delete a file on";:GOSUB10000
4010 GOSUB50000:PRINT"Type in the name of the file that you";
4015 PRINT" want to delete":PRINT
4016 INPUT"and depress RETURN (1-6 Characters) ";A$
4017 IFLEN(A$)>6THEN4010
4018 IFA$=""THENRUN
4020 IFLEN(A$)<6THENA$=A$+" ":GOTO4020
4030 S=5:GOSUB20000:IFE$="N"THEN40000
4040 S=4:GOTO20000
5000 PRINT"Data disk create utility":PRINT:PRINT
5010 PRINT"Be sure the tutorial disk is in drive A":PRINT
5020 GOSUB10200:DISK!"SE A":DISK!"CA 5C00=11,2"
5030 DISK!"CA 5D00=11,3":DISK!"CA 5E00=11,4":DISK!"CA 5F00=11,5"
5033 IFS=6THENDISK!"CA 5C00=11,6"
5035 GOSUB5100:DISK!"GO 2768":DISK!"SA 12,1=5C00/1"
5040 DISK!"SA 12,2=5D00/1":DISK!"SA 12,3=5E00/1"
```

```
5050 DISK!"SA 12,4=5F00/1":IFS=6THENGOSUB6000
5070 GOSUB50010
5080 PRINT"Your diskette is now ready for data files.":PRINT
5090 PRINT:GOTO5505
5100 GOSUB50010
5105 PRINT"Remove your tutorial diskette from drive A and":PRINT
5110 PRINT"replace it with your blank diskette.":PRINT:GOTO10200
5500 GOSUB50010
5505 PRINT"Remove your blank diskette from drive A and":PRINT
5510 PRINT"replace it with your tutorial diskette.":PRINT:GOTO10200
6000 P$="8":T1=1:T2=10:GOSUB2153
6010 T1=13:T2=27:GOTO2153
7000 L=12926:IFPEEK(8999)=58THENL=14974
7005 PRINT"Buffer set utility":PRINT:PRINT
7010 PRINT"Type in the number of file buffers";
7015 PRINT" you need":PRINT
7020 INPUT"and depress RETURN (0,1, or 2) <0>";N
7030 IFN<0ORN>2THENGOSUB50010:GOTO7010
7040 L=L+N*2048:N=N+1:GOSUB50010:ONNGOTO7100,7200,7300
7100 PRINT"No file buffers are resident.":GOTO7500
7200 PRINT"A single buffer is now resident.":GOTO7500
7300 PRINT"Two buffers are now resident."
7500 PRINT
7510 PRINT"Type in your program and save it on your diskette."
7520 GOSUB59000
7540 POKE120,L+1-INT((L+1)/256)*256:POKE121,INT((L+1)/256):POKEL,0:NEW
8000 X=PEEK(8960):POKE133,X:RUN"COPIER
10000 PRINT" which drive ?":PRINT
10004 INPUT"Type A,B,C or D and depress RETURN <A> ";D$
10005 IFD$=""THEND$="A"
10010 IFD$<"A"ORD$>"D"ORLEN(D$)<>1THENGOSUB50000:GOTO10004
10020 DISK!"SE "+D$:RETURN
10100 PRINT"Do you want to list the directory":PRINT
10105 INPUT"to the printer (Yes or No) <No> ";P$
10110 DV=2:IFLEFT$(P$,1)="Y"THENDV=1
10120 RETURN
10200 INPUT"Depress RETURN to continue ";P$:RETURN
20000 NF=0:E$="N":FORK=0TO39:AL%(K)=0:NEXT
20003 IFS=5THENPRINT:PRINT:PRINT"Please wait.";
20005 DISK!"CA 2E79=12,1":GOSUB 20100
20006 IFS>1ANDS<5THEN DISK!"SA 12,1=2E79/1
20008 IF S>1 AND E$="Y" THENRETURN
20010 DISK!"CA 2E79=12,2":GOSUB20100
20015 IFS>1ANDS<5THEN DISK!"SA 12,2=2E79/1
20020 RETURN
20100 FORI=11897TO12145STEP8
20110 ONSGOSUB20200,30000,20300,20400,30000
20115 IFNF=1ANDS=2THENGOSUB30300:E$="Y":RETURN
20120 NEXT:RETURN
20200 IFPEEK(I)=35THENNF=NF+1:RETURN
20210 GOSUB30100
20220 PRINT#DV,N$;TAB(12);FNA(PEEK(I+6));TAB(16);"-";
20230 PRINT#DV,TAB(17);FNA(PEEK(I+7)):RETURN
20300 GOSUB30100:IFD$<>N$THENRETURN
20310 E$="Y":GOTO30200
20400 GOSUB30100:IFA$<>N$THENRETURN
20410 E$="Y":A$="######":GOSUB30200:POKEI+6,0:POKEI+7,0:RETURN
30000 IFPEEK(I)=35THENNF=NF+1:RETURN
```

```
30005 GOSUB30100:IFN$=A$THENE$="Y"
30010 T0=FNA(PEEK(I+6)):T9=FNA(PEEK(I+7))
30020 FORK=T0T0T9:AL%(K)=-1:NEXT:RETURN
30100 N$="":FORJ=ITOI+5:N$=N$+CHR$(PEEK(J)):NEXT
30110 PRINT" ";:RETURN
30200 FORJ=1TO6:POKEI+J-1,ASC(MID$(A$,J,1)):NEXT:RETURN
30300 GOSUB30200:POKEI+6,FNB(T1):POKEI+7,FNB(T2):NF=255:RETURN
40000 PRINT:PRINT:PRINT"** "CHR$(34);A$;CHR$(34)" was not found ";
40010 PRINT"in the directory. **":PRINT:GOTO10200
40100 PRINT:PRINT:PRINT"** "CHR$(34);A$;CHR$(34)" already exists ";
40110 PRINT"in the directory. **":PRINT:GOTO10200
40200 PRINT:PRINT:PRINT"** Directory full **":PRINT:GOTO10200
50000 ST=11984:FORII=0TO36:READSC:POKEST+II,SC:NEXT:RESTORE     ST=$2ED0
50010 IFPEEK(8999)=58THEN PRINTCHR$(27);CHR$(21):RETURN
50015 POKE8955,208
50020 POKE8956,46:X=USR(X):RETURN
50030 DATA 169,208,141,219,46,169,32,162,0,157,0,208,232
50040 DATA 208,250,172,219,46,200,140,219,46,192,232,240,10
50050 DATA 192,216,208,235,160,224,169,14,208,239,96
59000 POKE741,76:POKE750,78:POKE2073,173:POKE2893,55:POKE2894,8
59010 POKE2888,27:X=PEEK(8960):POKE133,X
59020 RETURN
60000 GOSUB59000
60010 GOSUB50000:CLEAR
60020 PRINT"The system is now open for modification."
```

COPIER

COPIER is a routine which can be used for copying diskettes. The method you should use depends upon the configuration of your system. If your system has only one disk drive, then you can select the single drive copier automatically by copying from drive A to drive A. If your system has a dual disk drive then you can select the dual drive copier by selecting any other combination of drives.

The first step in using either of the copiers is to select two disks: the disk you wish to copy FROM and the disk you wish to copy TO. (Note: It is possible to copy onto a previously used diskette. The diskette is initialized as part of the copy process.) Once you have selected the two diskettes to be used, carefully follow the set of instructions given below. Type

RUN"COPIER"

The following instructions will be displayed on the screen (each instruction will be displayed after you have entered an answer to the previous one):

```
- Diskette copier -
Copy from which drive (A/B/C/D) ? _
Copy to which drive (A/B/C/D) ? _
What is the last track to be copied (Inclusive) <Ø-39> ? _
Are you ready to start copying (Y/N) ? _
```

The only difference between the single drive copier and the dual drive copier is that the single drive copier prompts you to insert the master and blank diskette during the copy operation.

A sequence of numbers will be displayed which shows the progression of the copy process through the tracks. When the copy process is completed, you will be asked to replace the OS-65D disk 5 in the disk drive. When you confirm that you have done so, the menu will again be displayed on the screen.

File name : COPIER

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 17	Track 13
Length :	2 Tracks	2 Tracks
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	11 PAGE OBJECT FILE	12 PAGE OBJECT FILE

Mini-floppy to 8" floppy program conversions

NOT APPLICABLE

CHANGE

The changes in system parameters that can be effected by running CHANGE are:

Workspace Limits

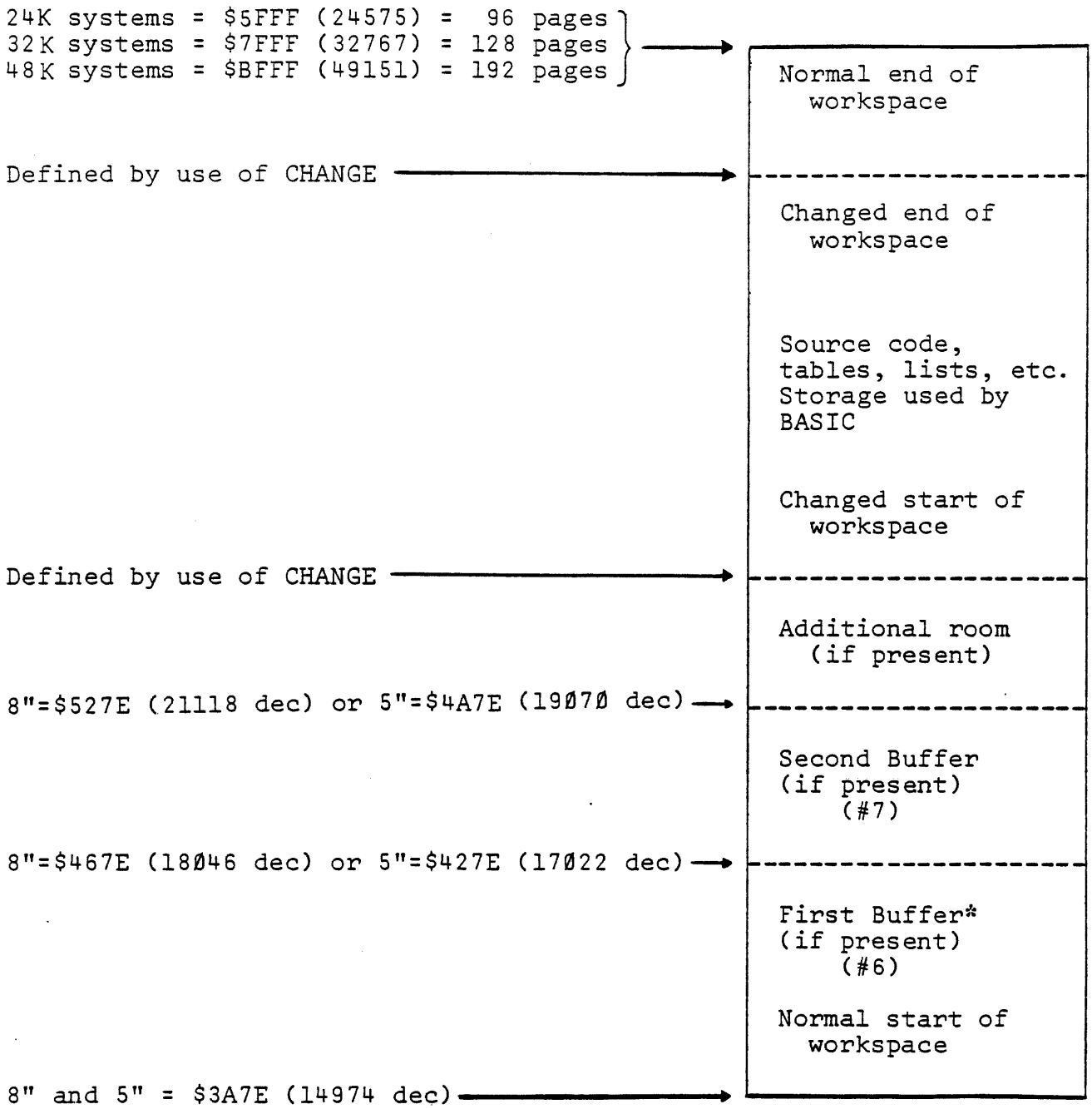
1. For creating buffers for disk input and output.
2. For leaving room for special programs such as machine language subroutines.

The "workspace" is that RAM area where the assembler and BASIC source programs reside. It is used to hold these source programs and various tables, lists, etc., that are used during assembly or BASIC program interpretation. The workspace normally begins at 14974 (hex 3A7E) for full size floppy disk systems and mini-floppy disk systems under OS-65D V3.3. Under other versions of OS-65D, the workspace normally begins at 12670 (hex 317E) for full size floppy disk systems and at 12926 (hex 327E) for mini-floppy disk systems.

The end of the workspace is normally the end of the main memory (that memory which starts at address zero and is contiguous up to some higher address). For 24K systems that means $5FFF (24575); 32K systems are $7FFF (32767); and, 48K configurations have the end of workspace at $BFFF (49151). See the next page for a diagramatic summary of the workspace.

Some instances where it may be necessary to change the workspace limits are:

1. When the program to be entered will use disk data files, that is, when PRINT#6, PRINT#7, INPUT#6 or INPUT#7 are used in the program. In this case, it is necessary to reserve space in the low part of memory to be used as disk I/O buffers. For example, if a PRINT#6 statement is executed then the data is sent to the buffer in memory,



* 8" buffer size = 3072 ($C00) bytes or 12 pages
(equivalent to one disk track)

5" buffer size = 2048 ($800) bytes or 8 pages
(equivalent to one disk track)

(Compare to the diagram in Chapter 6)

not directly to the disk. The data is not put on the disk until the execution of a DISK PUT at which time the entire buffer area is written onto the disk. The user should note that CHANGE cannot be used to put a buffer on a program that already exists on disk or in memory. The user should also note that a more convenient method to create a buffer for a new program is to load a program with the correct number of buffers from the disk (assuming the user has such a program) and then type NEW.

2. If the user wishes to write a BASIC program that calls a machine language subroutine, then space may be reserved for the subroutine using CHANGE. The user should note that the high part of memory may not be safe for this purpose (without using CHANGE) even if the BASIC program in the workspace is very short. The reason is that when a BASIC program is running, string variables are stored at the end of the workspace. CHANGE may be used to leave space at the beginning of the workspace, or to set the end of the workspace before the physical end of memory so as to leave a part of memory unaffected by the BASIC program. Putting machine language subroutines at the beginning of the workspace may be the more convenient place because anything in the part of memory between the start of the user's workspace and the beginning of the BASIC program is saved along with the BASIC program when a

```
DISK!"PUT..."
```

is executed, but the code above the adjusted end of memory is not. A machine language program should not be put between the start of the user's workspace and the beginning of the BASIC program if the BASIC program uses disk data files because the machine language program will be overwritten during disk access from file I/O.

The computer uses five memory locations to store the workspace limit parameters.

1. The number of pages in memory is stored at 8960 (=$2300) in the DOS kernel. This is a one byte parameter. For example, at boot up a value is automatically placed in this location reflecting the actual memory size. The second series of questions (after terminal width questions) asked by CHANGE allow the user to change this parameter. The value stored at this location is used except when the DOS commands

```
BASIC
      or
ASSM
      or
EM
```

are entered in response to the A*prompt.

2. BASIC stores the address for the end of the workspace on page Ø, memory locations 132 (=$84) and 133 (=$85). The low half or low byte of this address is stored at 132 and the high half or high byte at 133. For example, on a 24K computer the end of memory would be $5FFF, so the contents of the low byte 132 would be $FF (255) and the contents of 133 would be $5F (95). When the DOS command

BASIC

is entered in response to the A* prompt, the value at 896Ø is used to compute a value to be stored at 132 and 133. When the value stored at these locations is changed by CHANGE, the effect on BASIC will be immediate. For example, this value is used when BASIC makes an "out of memory" check.

3. Memory locations 121 (=$79) and 122 (=$7A) contain the address of the beginning of BASIC programs in the workspace. The low byte of this address is stored at 121 and the high byte at 122. When the system boots up, the contents are generally $7E (126) and $3A (58) in 121 and 122, respectively. When this value is changed by CHANGE, the byte at the beginning of workspace is POKEd with Ø.

Important: See page 147 for a warning on the use of the workspace after is has been modified by CHANGE.

The following directions explain how to change systems parameters using the CHANGE program.

The program output and your inputs are shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input. Enter RUN"CHANGE". You will see:

```
CHANGE PARAMETER UTILITY

CHANGE BASIC'S WORKSPACE LIMITS (Y/N)?
```

Enter YES or NO. If you enter NO, the program terminates. If you enter YES, the program requests the following:

```
HOW MANY 12 (8 for mini-floppy) PAGE BUFFERS DO
YOU WANT BEFORE THE WORKSPACE (Ø, 1, or 2)?
```

Enter Ø, 1, or 2 to reserve that many track buffers at the beginning of the workspace. Note that device 6 memory buffer I/O uses the first buffer while device 7 uses the second buffer. If

the answer is YES then a new value is POKEd to 132 and 133. If no buffers are specified, the program asks:

WANT TO LEAVE ANY ROOM BEFORE THE WORKSPACE (Y/N)?

Enter YES or NO. If you enter NO, the program outputs the address of the start of the BASIC workspace as shown below. IF YES is entered, proceed to the "HOW MANY BYTES?" question below. An answer of YES will change the value at 132 and 133.

If one or more buffers was specified, the program continues with:

WANT TO LEAVE ANY ADDITIONAL ROOM (Y/N)?

Enter YES or NO. If you enter YES, the following question is asked:

HOW MANY BYTES?

Enter the number of additional bytes to be allocated before the start of the workspace.

The program then outputs the new address for the start of the workspace and the total number of bytes reserved for buffers, etc.

```
THE BASIC WORKSPACE WILL BE SET TO START AT aaaaa
LEAVING bbbb BYTES FREE IN FRONT OF THE WORKSPACE

IS THAT ALRIGHT (Y/N)?
```

Enter YES or NO. If you enter NO, the program requests that you specify an exact lower limit address for the workspace.

NEW LOWER LIMIT?

Enter a lower limit address. The program then confirms this value by outputting:

bbbb BYTES WILL BE FREE BEFORE THE WORKSPACE

The program then continues with:

YOU HAVE xx K OF RAM

DO YOU WANT TO LEAVE ANY ROOM AT THE TOP (Y/N)?

Enter YES or NO. A YES answer will cause values to be POKEd to 121 and 122. If you enter YES, the following question is asked:

HOW MANY BYTES?

Enter the number of bytes to be allocated between the top of the workspace and the end of main memory.

The program then outputs:

THE BASIC WORKSPACE WILL BE SET TO END AT ccccc
LEAVING dddd BYTES FREE AFTER THE WORKSPACE

IS THAT ALRIGHT (Y/N)?

Enter YES or NO. If you enter NO, the program requests that you specify an exact number limit address for the workspace.

NEW UPPER LIMIT?

Enter an upper limit address. The program then confirms this value by outputting:

eeee BYTES WILL BE FREE AFTER THE WORKSPACE

Note that the reservation of space after the workspace is not recorded on disk with a program when it is saved in a file. The allocation is only recorded as a RAM resident change to the BASIC interpreter and remains in effect until explicitly changed again, or BASIC is reloaded by typing BAS in the DOS command mode. Later, running a program that results in an "Out of Memory" (OM) error may be the result of a workspace that has been unnecessarily left reduced.

Program output continues with:

```
YOU WILL HAVE fffff BYTES FREE IN THE WORKSPACE
IS THAT ALRIGHT (Y/N)?
```

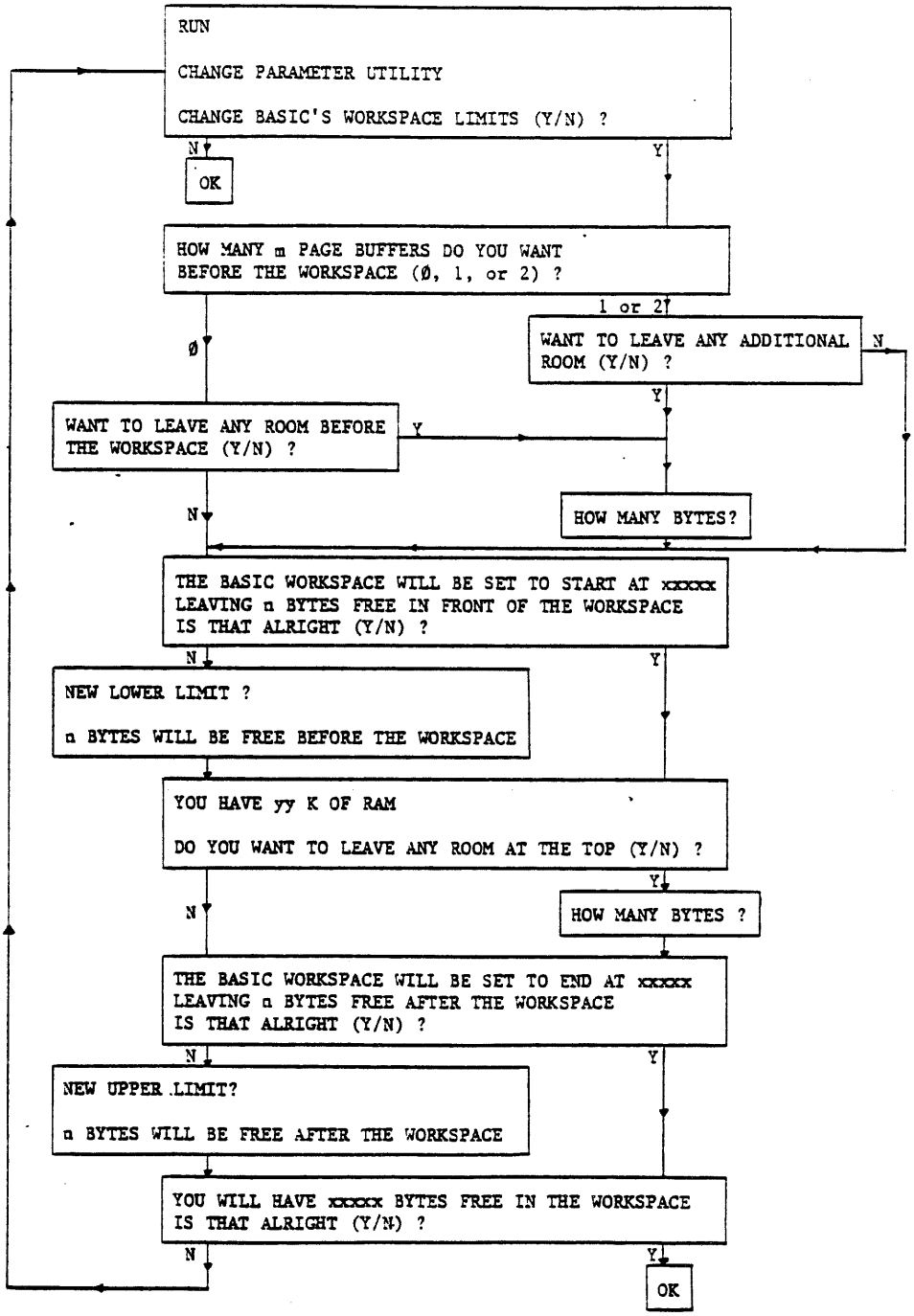
Enter YES or NO. If NO is entered, the Change Parameter Utility Program restarts from the beginning. Otherwise, the requested changes are made, the workspace contents are cleared and the program terminates.

WARNING: If you enter a DISK!"LOAD filename" command after running CHANGE the workspace parameters are not those selected by CHANGE, but rather those associated with "filename".

Therefore, if you are not ready to enter a new program in your CHANGE modified workspace, you should enter a short dummy program, perhaps with a few REM statements specifying the special workspace parameters. Then save this with a DISK!"PUT filename" command. Later, this file can be LOADed and the special workspace parameters will be automatically set. Then you can enter a new program or merge in an old program via indirect files (see the BASIC Reference Manual, Chapter 12) without alterning the workspace specififcations.

For a discussion on how to use a machine language subroutine in your CHANGEd workspace, see Appendix 11.

CHANGE - Logical Flowchart



File name : CHANGE

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Tracks 19-2Ø	Track 15
Length :	2 Tracks	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

```
25 XZ=14974:IF PEEK(8999)=49 THEN XZ=12670

40 PRINT "HOW MANY 12 PAGE BUFFERS DO YOU WANT"

60 L=XZ+B*3072:REM L=$317E PLUS B*$C00
```

```
1 PRINTCHR$(27);CHR$(28);
10 REM CHANGE PARAMETER UTILITY UNDER OS-65D VERSION 3.0
20 REM *R1 1/5/79
25 XZ=14974:IFPEEK(8999)=50THENXZ=12926
30 PRINT : PRINT "CHANGE PARAMETER UTILITY" : PRINT
36 PRINT : INPUT "CHANGE BASIC'S WORKSPACE LIMITS (Y/N) ";A$
37 PRINT:IF A$<>"Y" THEN END
40 PRINT"HOW MANY 8 PAGE BUFFERS DO YOU WANT "
41 INPUT"BEFORE THE WORKSPACE (0, 1, or 2) ";B
45 PRINT
50 IFB<0ORB>2ORB<>INT(B)THENPRINT"ANSWER 0, 1, OR 2":PRINT:GOTO40
60 L=XZ+B*2048 : REM L=$327E PLUS B*$800
70 IF B<>0 THEN 130
80 INPUT "WANT TO LEAVE ANY ROOM BEFORE THE WORKSPACE (Y/N) ";A$
90 IF MID$(A$,1,1)<>"Y" THEN 170
100 PRINT:INPUT "HOW MANY BYTES ";B
110 L=L+B
120 GOTO 170
130 INPUT "WANT TO LEAVE ANY ADDITIONAL ROOM (Y/N) ";A$
140 IF MID$(A$,1,1)<>"Y" THEN 170
150 PRINT:INPUT "HOW MANY BYTES ";B
160 L=L+B
170 PRINT:PRINT "THE BASIC WORKSPACE WILL BE SET TO START AT";L
180 PRINT "LEAVING";L-XZ;"BYTES FREE IN FRONT OF THE WORKSPACE"
190 INPUT "IS THAT ALRIGHT (Y/N) ";A$
200 IF MID$(A$,1,1)="Y" THEN 210
204 PRINT:INPUT "NEW LOWER LIMIT ";L:PRINT:IF L<XZ THEN 204
206 PRINT L-XZ;"BYTES WILL BE FREE BEFORE THE WORKSPACE"
210 MP=PEEK(8960)
220 PRINT:PRINT "YOU HAVE";(MP+1)/4;"K OF RAM":PRINT
230 U=(MP+1)*256
240 INPUT "DO YOU WANT TO LEAVE ANY ROOM AT THE TOP (Y/N) ";A$
250 IF MID$(A$,1,1)<>"Y" THEN 280
260 PRINT:INPUT "HOW MANY BYTES ";B
270 U=U-B
280 PRINT:PRINT "THE BASIC WORKSPACE WILL BE SET TO END AT";U
290 PRINT "LEAVING";(MP+1)*256-U;"BYTES FREE AFTER THE WORKSPACE"
300 INPUT "IS THAT ALRIGHT (Y/N) ";A$
310 PRINT:IF MID$(A$,1,1)="Y" THEN 320
312 INPUT"NEW UPPER LIMIT";U:PRINT:IFU>49152ORU<L+3THEN312
314 PRINT (MP+1)*256-U;"BYTES WILL BE FREE AFTER THE WORKSPACE"
320 PRINT:PRINT"YOU WILL HAVE";U-L+1;"BYTES FREE IN THE WORKSPACE"
330 INPUT "IS THAT ALRIGHT (Y/N) ";A$
340 IF MID$(A$,1,1)<>"Y" THEN RUN
350 REM
360 REM NOW DO THE ADJUSTMENTS
370 REM
380 POKE132,U-INT(U/256)*256:POKE133,INT(U/256)
390 POKE120,L+1-INT((L+1)/256)*256:POKE121,INT((L+1)/256):POKEL,0:NEW
1000 REM
1010 REM
1020 PRINT "THE TERMINAL WIDTH IS SET FOR";PEEK(23)
1030 INPUT "DO YOU WANT TO CHANGE IT (Y/N)";A$
1040 IF A$<>"Y" THEN 1100
1050 INPUT "NEW VALUE";WD
1060 IF WD<14 OR WD>255 THEN PRINT "BAD VALUE" : GOTO 1050
1070 POKE 23,WD
1080 NC=INT(WD/14)*14: REM *R1
1090 POKE 24,NC
```

```
1100 PRINT : PRINT "BASIC & THE ASSEMBLER USE";(PEEK(8960)+1)/4;
1110 PRINT"K OF MEMORY (TOTAL) (";PEEK(8960)+1;"PAGES )":REM*R1
1120 INPUT "WOULD YOU LIKE TO CHANGE THIS (Y/N)";A$
1130 IF A$<>"Y" THEN 1170
1140 INPUT "HOW MANY PAGES SHOULD THEY USE (MIN: 52) ";KK
1150 IF KK<52 GOTO 1140: REM *R1
1152 IF KK>192 THEN PRINT"ABSOLUTE MAX IS 192":GOTO 1140:REM*R1
1154 DD=PEEK(KK*256-1): POKE KK*256-1,256-DD
1156 ND=PEEK(KK*256-1):POKE KK*256-1,DD: IF ND=256-DD GOTO 1160
1158 PRINT "THIS COMPUTER DOESN'T HAVE";KK;"PAGES OF MEMORY": GOTO 1140
1160 POKE 8960,KK-1: REM *R1
1170 END
```

CREATE

This utility program is used to create new named files. Note that a file must have been created with this program before it can be referenced by any of the file commands. To create a file, type:

```
RUN "CREATE"
```

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

```
FILE CREATION UTILITY
PASSWORD?
```

Enter PASS.

The program continues with an explanation of its operation:

```
CREATES AN ENTRY IN DIRECTORY FOR A NEW FILE AND
INITIALIZES THE TRACKS THAT THE NEW FILE WILL
RESIDE ON.  THE TRACKS WILL CONTAIN NULLS WITH
A RETURN AT THE END OF THE TRACK.

FILE NAME?
```

Enter a one to six character file name that is not a duplicate of an existing file name. It must start with a letter.

```
FIRST TRACK OF FILE?
```

Enter the number of the first track the file is to reside on. Note that a file always begins on a track boundary and resides on a whole number of tracks.

```
NUMBER OF TRACKS IN FILE?
```

Enter the number of tracks on which the file is to reside. All tracks assigned to a file must not have been previously assigned. It is often useful to run a directory of the diskette prior to running the create utility so that the free tracks on the diskette can be identified.

The program then continues with:

```
12 (8 for mini-floppy) PAGES PER TRACK.  IS THIS OK?
```

Type YES if the specified number of pages per track is acceptable; otherwise, type NO. If you type NO, the following question is asked:

HOW MANY PAGES PER TRACK THEN?

Enter the number of pages of storage you want each track to contain. Any number up to the default number of pages is acceptable. For full size diskettes this is twelve pages and for mini-diskettes it is eight pages per track.

The file will now be created and its name and track location will be entered into the directory. Each of the tracks of the file will be initialized to nulls with a return character at the end of each track.

File name : CREATE

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 21	Track 16
Length :	2 Tracks	2 Tracks
Buffers for:		
Device #6:	YES	YES
Device #7:	NO	NO
Other :	18 BYTE OBJECT FILE	18 BYTE OBJECT FILE

Mini-floppy to 8" floppy program conversions

```
230 DATA 160,0,162,12,152,153,126,58,200,208,250

490 DIM AL%(76)

500 FORI=0TO76:AL%(I)=0:NEXT

510 DISK!"CA 2E79=08,2":GOSUB10000

520 DISK!"CA 2E79=08,1":GOSUB10000

570 IF FL=1 THEN DISK!"SA 08,1=2E79/1":END

580 DISK!"CA 2E79=08,1":GOSUB20000

590 IF FL=1 THEN DISK!"SA 082=2E79/1":END

20090 IF T0<8 OR T0>76 THEN 20080

20110 IF NT<1 OR NT+T0>77 THEN 20100

20170 PG=12

20180 INPUT"12 PAGES PER TRACK. IS THIS OK";B$

20205 IF PG<1 OR PG>12 GOTO20200

20210 IF PG<>12 THEN ?:?"DEFAULTS SET FOR 12 PAGES"
```

```
230 DATA160,0,162,8,152,153,126,58,200,208,250
240 DATA238,133,46,202,208,244,96
250 FORI=11902TO11919:READD:POKEI,D:NEXT
260 IFPEEK(8999)=50THENPOKE11909,50
280 POKE8955,126:POKE8956,46:X=USR(X):POKE8955,212:POKE8956,34
320 PRINT:PRINT"FILE CREATION UTILITY":PRINT
330 PN=11897
335 INPUT"PASSWORD";A$:IFA$<>"PASS"THENEND
337 PRINT
340 DEFFNA(X)=16*INT(X/10)+X-10*INT(X/10)
350 DEFFNB(X)=10*INT(X/16)+X-16*INT(X/16)
360 DATA1,2,3,4,5,6,7,8,9,A,B,C
370 PRINT"CREATES AN ENTRY IN DIRECTORY FOR A NEW FILE"
380 PRINT"AND INITIALIZES THE TRACKS THAT THE NEW FILE WILL"
390 PRINT"RESIDE ON.  THE TRACKS WILL CONTAIN NULLS WITH A"
400 PRINT"RETURN AT THE END OF THE TRACK."
410 PRINT
420 INPUT"FILE NAME";A$
430 IFLEN(A$)>6THEN410
440 IFLEN(A$)<6THENA$=A$+" ":GOTO440
450 IFMID$(A$,1,1)<"A"ORMID$(A$,1,1)>"Z"THEN420
460 REM
490 DIMAL%(39)
500 FORI=0TO39:AL%(I)=0:NEXT
510 DISK!"CA 2E79=12,2":GOSUB10000
520 DISK!"CA 2E79=12,1":GOSUB10000
530 REM
540 REM
560 GOSUB20000
570 IFFL=1THENDISK!"SA 12,1=2E79/1":END
580 DISK!"CA 2E79=12,2":GOSUB20000
590 IFFL=1THENDISK!"SA 12,2=2E79/1":END
600 PRINT"** DIRECTORY FULL **":END
10000 REM
10010 REM
10030 FORI=PN+6TOPN+254STEP8
10040 IFPEEK(I-6)=35THEN10100
10050 C$="":FORK=1TO6:C$=C$+CHR$(PEEK(I-7+K)):NEXTK:IFC$<>A$THEN10090
10080 PRINT"** FILE NAME ";CHR$(34);A$;CHR$(34);" IN USE **":RUN
10090 T0=FNB(PEEK(I)):T9=FNB(PEEK(I+1))
10095 FORK=T0TOT9:AL%(K)=-1:NEXTK
10100 NEXTI
10110 RETURN
20000 REM
20010 REM
20030 FORI=PNTOPN+248STEP8
20040 IFPEEK(I)=35THEN20080
20050 NEXTI
20060 FL=0:RETURN
20080 PRINT:INPUT"FIRST TRACK OF FILE";T0
20090 IFT0<13ORT0>39THEN20080
20100 INPUT"NUMBER OF TRACKS IN FILE";NT
20110 IFNT<1ORNT+T0>40THEN20100
20115 T9=T0+NT-1
20120 FK=0
20130 FORK=T0TOT9
20140 IFAL%(K)THENPRINT"** TRACK";K;"IN USE **":FK=1
20150 NEXTK
```

```
20160 IFFK<>0THENRUN
20170 PG=8
20180 INPUT"8 PAGES PER TRACK.  IS THIS OK";B$
20190 IFMID$(B$,1,1)="Y"THEN20220
20200 INPUT"HOW MANY PAGES PER TRACK THEN";PG
20205 IFPG<1ORPG>8GOTO20200
20210 IFPG<>8THENPRINT:PRINT"NOTE: ALL DEFAULTS ARE SET FOR 8 PAGES!"
20220 FORJ=0TO5:POKEI+J,ASC(MID$(A$,J+1,1)):NEXTJ
20250 POKEI+6,FNA(T0):POKEI+7,FNA(T9)
20260 FORI=1TOPG:READP$:NEXTI
20270 FORI=T0TOT9
20280 T$=RIGHT$(STR$(I+100),2)
20285 J$="3A7E":IFPEEK(8999)=50THENJ$="327E"
20290 DISK!"IN "+T$:DISK!"SA "+T$+",1="+J$+"/"+P$
20310 NEXTI
20320 FL=1:RETURN
```

DELETE

This utility program may be used to delete a named file from the directory. DELETE frees the tracks on which that file resided, but it does not actually alter the contents of those tracks. Consequently, until a new file is created residing on those tracks or the tracks are otherwise changed, the contents of the old (deleted) file are still recoverable by a direct track number access. To delete a named file, type:

```
RUN "DELETE"
```

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

```
DELETE UTILITY
REMOVES AN ENTRY FROM THE DIRECTORY
PASSWORD?
```

Enter PASS. The program continues with:

```
FILE NAME?
```

Enter the name of the file to be deleted. The file will now be deleted from the directory.

File name : DELETE

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 23	Track 18
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

```
5Ø DISK!"CA 2E79=Ø8,1"
6Ø GOSUB1ØØØØ:IF FL=1 THEN DISK!"SA Ø8,1=2E79/1":END
7Ø DISK!"CA 2E79=Ø8,2"
8Ø GOSUB1ØØØØ:IF FL=1 THEN DISK!"SA Ø8,2=2E79/1":END
```

```
10 REM DELETE FILE UTILITY UNDER OS-65D VERSION 3.0
20 REM
30 PRINT:PRINT"DELETE UTILITY":PRINT
32 PRINT "REMOVES AN ENTRY FROM THE DIRECTORY":PRINT
33 INPUT "PASSWORD";A$ : IF A$<>"PASS" THEN END
34 PRINT
35 FLAG=0 : PN=11897
40 INPUT "FILE NAME";A$
45 IF LEN(A$)>6 THEN 40
47 IF LEN(A$)<6 THEN A$=A$+" " : GOTO 47
50 DISK ! "CA 2E79=12,1"
60 GOSUB 10000 : IF FL=1 THEN DISK ! "SA 12,1=2E79/1" : END
70 DISK ! "CA 2E79=12,2"
80 GOSUB 10000 : IF FL=1 THEN DISK ! "SA 12,2=2E79/1" : END
90 PRINT "** ";CHR$(34);A$;CHR$(34);" NOT FOUND IN DIRECTORY **":PRINT
100 END
10000 REM
10010 REM SEE IF FILE NAME A$ IS IN DIRECTORY BUFFER
10020 REM
10030 FOR I=PNT TO PN+248 STEP 8
10040 FOR J=I TO I+5
10050 IFPEEK(J)<>ASC(MID$(A$,J-I+1,1)) THEN 10100
10060 NEXT J
10070 FOR J=I TO I+5 : POKE J,ASC("#") : NEXT J
10080 POKE I+6,0 : POKE I+7,0
10090 FLAG=1 : RETURN
10100 NEXT I
10110 FLAG=0 : RETURN
```

DIR

This utility program may be used to display a list of all currently existing named files and the numbers of the tracks on which they reside. The directory can be unsorted or sorted in alpha numeric order by file name or sequentially by track number. To display a directory, type:

```
RUN "DIR"
```

The program will display a title and a menu followed by a prompt for information as shown below. Any unacceptable response will cause an error message and/or a repeat of the request for information. If an error message is encountered, then type RUN to restart the utility.

```
DIRECTORY UTILITY

1> Directory
2> Directory sorted by name
3> Directory sorted by track

Type 1, 2 or 3 and depress RETURN ?
```

Enter 2 or 3 to specify a named or a track sort, respectively. The program continues with:

```
LIST ON LINE PRINTER INSTEAD OF DEVICE #d?
```

Enter YES or NO. (d is the current output device assignment.)

If you enter YES, the directory output will be on device 1; otherwise, it will be on the currently assigned output device, the video monitor. If you answer YES and there is no device 1 on the system, the directory will not be displayed. The system will lock up and you will have to reboot.

If the number 1 was entered to the menu prompt above,

THEN IT WILL BE UNSORTED

is displayed and the directory list will be in the same order as the actual entries in the directory.

Sample mini-floppy directory displays, sorted by name and track number, appear on the next page. The line at the bottom of each directory stating, "46 ENTRIES FREE OUT OF 64", means that eighteen directory files use up eighteen of the 64 available directory entires. Forty-six entries remain free for new file names.

OS-65D VERSION 3.3

-- DIRECTORY --

FILE NAME	TRACK RANGE
ASAMPL	34 - 34
ATNENB	35 - 35
BEXEC*	14 - 16
CHANGE	19 - 20
COLORS	36 - 36
COMPAR	39 - 39
COPIER	17 - 18
CREATE	21 - 22
DELETE	23 - 23
DIR	24 - 24
MODEM	37 - 38
OS65D3	0 - 13
RANLST	25 - 26
RENAME	27 - 27
SECDIR	28 - 28
SEQLST	29 - 30
TRACE	31 - 31
ZERO	32 - 33

46 ENTRIES FREE OUT OF 64

OS-65D VERSION 3.3

-- DIRECTORY --

FILE NAME	TRACK RANGE
OS65D3	0 - 13
BEXEC*	14 - 16
COPIER	17 - 18
CHANGE	19 - 20
CREATE	21 - 22
DELETE	23 - 23
DIR	24 - 24
RANLST	25 - 26
RENAME	27 - 27
SECDIR	28 - 28
SEQLST	29 - 30
TRACE	31 - 31
ZERO	32 - 33
ASAMPL	34 - 34
ATNENB	35 - 35
COLORS	36 - 36
MODEM	37 - 38
COMPAR	39 - 39

46 ENTRIES FREE OUT OF 64

File name : DIR

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 24	Track 19
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

```
1ØØ6Ø DISK! "CA 2E79=Ø8,1"

1ØØ8Ø DISK! "CA 2E79=Ø8,2"
```

```
1 PRINTCHR$(27);CHR$(28);
10 REM
20 REM
30 NF=0
40 PN=11897
50 DEF FNA(X)=10*INT(X/16)+X-16*INT(X/16)
60 DIM NM$(64),T0%(64),T9%(64)
70 AV=0
80 DV=1 : Y=1 : X=PEEK(8994)
82 IF X=<Y THEN 86
84 DV=DV+1 : Y=Y+Y : GOTO 82
86 PRINT:PRINT "DIRECTORY UTILITY":PRINT
87 PRINT" 1> Directory":PRINT" 2> Directory sorted by name"
88 PRINT" 3> Directory sorted by track":PRINT
90 INPUT"Type 1, 2, or 3 and depress RETURN ";Z$
91 IFZ$<>"1"ANDZ$<>"2"ANDZ$<>"3"THENRUN
92 PRINT:IFZ$="2"THENZ$="N"
93 IFZ$="3"THENZ$="T"
95 PRINT "LIST ON LINEPRINTER INSTEAD OF DEVICE #";DV;
96 INPUT A$ : IF MID$(A$,1,1)="Y" THEN DV=1
100 IF Z$="N" OR Z$="T" THEN 10000
110 PRINT "THEN IT WILL BE UNSORTED"
10000 REM
10010 REM
10020 REM
10030 PRINT #DV : PRINT #DV, "OS-65D VERSION 3.3"
10035 PRINT #DV," -- DIRECTORY --" : PRINT #DV
10040 PRINT #DV,"FILE NAME    TRACK RANGE"
10050 PRINT #DV,"-------------------------"
10060 DISK ! "CA 2E79=12,1
10070 GOSUB 11000
10080 DISK ! "CA 2E79=12,2
10090 GOSUB 11000
10095 IF Z$="N" THEN GOSUB 20000
10097 IF Z$="T" THEN GOSUB 21000
10100 FOR I=0 TO AV-1
10110 PRINT #DV,NM$(I);TAB(12);T0%(I);TAB(16);"-";TAB(17);T9%(I)
10120 NEXT I
10130 PRINT #DV : PRINT #DV,64-AV;"ENTRIES FREE OUT OF 64" : PRINT #DV
10140 END
11000 REM
11010 REM
11020 REM
11040 FOR I=PN TO PN+248 STEP 8
11050 IF PEEK(I)=35 THEN 11130
11060 N$=CHR$(PEEK(I))+CHR$(PEEK(I+1))+CHR$(PEEK(I+2))+CHR$(PEEK(I+3))
11070 NM$(AV)=N$+CHR$(PEEK(I+4))+CHR$(PEEK(I+5))
11100 T0%(AV)=FNA(PEEK(I+6))
11110 T9%(AV)=FNA(PEEK(I+7))
11120 AV=AV+1
11130 NEXT I
11140 RETURN
20000 REM
20010 REM
20015 REM
20020 REM
20022 M=AV-1
20025 M=INT(M/2)
20030 IF M=0 THEN RETURN
```

```
20032 J=0 : K=AV-1-M
20040 I=J
20050 L=I+M
20070 IF NM$(I)<NM$(L) THEN 20120
20080 T$=NM$(I):NM$(I)=NM$(L):NM$(L)=T$
20090 T%=T0%(I):T0%(I)=T0%(L):T0%(L)=T%
20100 T%=T9%(I):T9%(I)=T9%(L):T9%(L)=T%
20105 I=I-M
20110 IF I>=0 THEN 20050
20120 J=J+1
20130 IF J>K THEN 20025
20140 GOTO 20040
21000 REM
21010 REM
21020 REM
21022 M=AV-1
21025 M=INT(M/2)
21030 IF M=0 THEN RETURN
21032 J=0 : K=AV-1-M
21040 I=J
21050 L=I+M
21070 IF T0%(I)<T0%(L) THEN 21120
21080 T$=NM$(I):NM$(I)=NM$(L):NM$(L)=T$
21090 T%=T0%(I):T0%(I)=T0%(L):T0%(L)=T%
21100 T%=T9%(I):T9%(I)=T9%(L):T9%(L)=T%
21105 I=I-M
21110 IF I>=0 THEN 21050
21120 J=J+1
21130 IF J>K THEN 21025
21140 GOTO 21040
```

RANLST

This utility program may be used to list the contents of a random access file either a single record at a time or in groups of contiguous records. The program assumes 128 byte records. To list a random file, type:

RUN "RANLST"

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

RANDOM ACCESS FILE READ

FILE NAME?

Enter the name of the random access file to be listed.

EXAMINE SINGLE RECORDS OR GROUPS (S/G)?

Enter S or G. If S is entered, the number of the single record to be listed is requested.

RECORD NUMBER?

Enter the number of the record to be listed. (Records are numbered from zero through n.) The specified record is listed, then the RECORD NUMBER questions is again asked. To terminate the program, merely depress RETURN in answer to this question.

If G is entered, above, the range of record numbers to be listed are requested.

FIRST RECORD?

Enter the number of the first record to be listed.

LAST RECORD?

Enter the number of the last record to be listed.

The specified records are listed, then the "SINGLE RECORDS OR GROUPS" question is again asked. To terminate the program, merely depress <RETURN>.

File name : RANLST

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 25	Track 2Ø
Length :	2 Tracks	2 Tracks
Buffers for:		
Device #6:	YES	YES
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

NONE

```
10 REM RANDOM ACCESS FILE LIST UTILITY UNDER OS-65D VERSION 3.0
20 REM
30 PRINT : PRINT "RANDOM ACCESS FILE READ" : PRINT
40 INPUT "FILE NAME";N$
50 IF LEN(N$)>6 THEN 40
70 DISK OPEN,6,N$
75 INPUT "EXAMINE SINGLE RECORDS OR GROUPS (S/G)";R$
77 IF R$="G" THEN 200
78 IF R$<>"S" THEN 75
80 PRINT : INPUT "RECORD NUMBER";R
90 DISK GET,R
100 INPUT #6,A$
110 PRINT : PRINT A$
120 GOTO 80
200 PRINT : INPUT "FIRST RECORD";R0
210 INPUT "LAST RECORD";R9
220 IF R9<R0 THEN 200
230 FOR R=R0 TO R9
240 DISK GET,R
250 INPUT #6,A$
260 PRINT : PRINT A$
270 NEXT R
280 GOTO 75
```

RENAME

This utility program may be used to change the name of any file listed in the directory. To rename a file, type:

```
RUN "RENAME"
```

The program will display a title and prompt you for information as shown below. Any unacceptable response will cause an error message and/or a repeat of the request for information.

```
RENAME UTILITY

OLD NAME?
```

Enter the name of the file to be renamed as it currently appears in the directory. The program then displays:

```
RENAME "aaaaaa" TO?  (aaaaaa is the old file name)
```

Enter the new name for this file (one to six characters, the first being a letter). This new name will then appear in the directory in place in "aaaaaa".

The name will be changed and the utility program will end with the BASIC prompt. For example,

```
RENAME UTILITY

OLD NAME? TEST
RENAME "TEST   " TO? TESTER

OK
```

File name : RENAME

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 27	Track 22
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

```
5Ø DISK! "CALL 2E79=Ø8,1"
6Ø GOSUB1ØØØØ:IF FLAG=1 THEN DISK!"SAVE Ø8,1=2E79/1":END
7Ø DISK! "CALL 2E79=Ø8,2"
8Ø GOSUB1ØØØØ:IF FLAG=1 THEN DISK!"SAVE Ø8,2=2E79/1":END
```

```
1 PRINTCHR$(27);CHR$(28);
10 REM RENAME FILE UTILITY UNDER OS-65D VERSION 3.0
20 REM
30 PRINT : PRINT "RENAME UTILITY" : PRINT
35 FLAG=0 : PN=11897
40 INPUT "OLD NAME";A$
45 IF LEN(A$)>6 THEN 40
47 IF LEN(A$)<6 THEN A$=A$+" " : GOTO 47
50 DISK ! "CA 2E79=12,1"
60 GOSUB 10000 : IF FL=1 THEN DISK ! "SA 12,1=2E79/1" : END
70 DISK ! "CA 2E79=12,2"
80 GOSUB 10000 : IF FL=1 THEN DISK ! "SA 12,2=2E79/1" : END
90 PRINT "** ";CHR$(34);A$;CHR$(34);" NOT FOUND IN DIRECTORY **"
100 END
10000 REM
10010 REM SEE IF FILE NAME A$ IS IN DIRECTORY BUFFER
10020 REM
10030 FOR I=PNT TO PN+248 STEP 8
10040 FOR J=I TO I+5
10050 IF CHR$(PEEK(J))<>MID$(A$,J-I+1,1) THEN 10100
10060 NEXT J
10070 PRINT "RENAME ";CHR$(34);A$;CHR$(34); : INPUT " TO";AN$
10075 IF LEN(AN$)>6 THEN 10070
10080 IF LEN(AN$)<6 THEN AN$=AN$+" " : GOTO 10080
10082 IF MID$(AN$,1,1)<"A" OR MID$(AN$,1,1)>"Z" THEN 10070
10085 FOR J=I TO I+5 : POKE J,ASC(MID$(AN$,J-I+1,1)) : NEXT J
10090 FLAG=1 : RETURN
10100 NEXT I
10110 FLAG=0 : RETURN
```

SECDIR

This utility program may be used to output the number and size of each sector on each of a specified range of tracks. To output a sector directory, type:

```
RUN "SECDIR"
```

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

```
SECDIR

USES OS-65D'S DIR COMMAND TO PRINT OUT A SECTOR MAP
OF A GIVEN RANGE OF TRACKS

FIRST TRACK?
```

Enter any valid track number greater than zero and less than the total number of existing tracks (76 for full size disks or 39 for mini-disks).

```
LAST TRACK?
```

Enter any valid track number greater than that entered for the first track.

A sector map for the specified tracks will be output, then the program will terminate. A sample of such is shown below.

```
SECTOR MAP DIRECTORY

 TRACK Ø1
 Ø1-Ø5
 Ø2-Ø5

TRACK Ø2
 Ø1-ØB
 etc.
 OK
```

In the sample, track 1 has two sectors, both five pages in length. Track 2 has one sector of 11 (hex B) pages.

File name : SECDIR

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 28	Track 23
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

```
4Ø IF TØ<1 OR TØ>76 THEN 3Ø

6Ø IF T9<TØ OR T9>76 THEN 5Ø
```

```
10 REM SECTOR DIRECTORY UTILITY UNDER OS-65D VERSION 3.0
15 REM
20 PRINT : PRINT "SECDIR" : PRINT
22 PRINT "USES OS-65D'S DIR COMMAND TO PRINT OUT A SECTOR"
24 PRINT "MAP OF A GIVEN RANGE OF TRACKS" : PRINT
30 PRINT : INPUT "FIRST TRACK";T0
40 IF T0<1 OR T0>39 THEN 30
50 PRINT : INPUT "LAST TRACK";T9
60 IF T9<T0 OR T9>39 THEN 50
70 PRINT : PRINT "SECTOR MAP DIRECTORY" : PRINT
80 FOR I=100+T0 TO 100+T9
90 DISK ! "DIR "+RIGHT$(STR$(I),2)
95 PRINT
100 NEXT I
110 END
```

SEQLST

This utility program may be used to list the contents of a sequential file. A sequential file is one in which all entries within the file are contiguous with no intervening gaps. To list a sequential file, type:

```
RUN "SEQLST"
```

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

```
SEQUENTIAL FILE LISTER
TYPE A CONTROL-C TO STOP
FILE NAME?
```

Enter the name of the sequential file to be listed.

The specified file is listed until you type a CONTROL-C or the end of the file is reached in which case the program terminates with the following end-of-file message:

```
ERR #D ERROR IN LINE 1ØØ
OK
```

File name : SEQLST

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 29	Track 24
Length :	2 Tracks	2 Tracks
Buffers for:		
Device #6:	YES	YES
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

NONE

```
10 REM SEQUENTIAL FILE LISTER UTILITY UNDER OS-65D VERSION 3.0
20 REM
30 PRINT : PRINT "SEQUENTIAL FILE LISTER" : PRINT
40 PRINT "TYPE A CONTROL-C TO STOP"
60 PRINT : INPUT "FILE NAME";A$
70 IF LEN(A$)>6 THEN 60
90 DISK OPEN,6,A$
100 INPUT #6,D$
110 PRINT D$
120 GOTO 100
```

TRACE

During the development of any new program, frequent testing is required to insure that the program is actually performing the desired tasks. The process of correcting the program problems (bugs) is called debugging. The TRACE utility is an aid for debugging BASIC programs, which displays the line number of each line of the BASIC program as it is being executed. This gives the programmer useful location information about any 'bugs' that he might be trying to cure.

This utility program may be used to enable or disable a line number trace for BASIC programs. To trace a BASIC program, type:

```
RUN "TRACE"
```

The program will display a title and then prompt you for information as shown below. Any unacceptable response will cause the request for information to be repeated.

```
TRACE UTILITY

WHEN BASIC'S TRACE FEATURE IS ENABLED, BASIC WILL PRINT OUT EACH
LINE NUMBER OF THE PROGRAM BEFORE IT IS EXECUTED.

ENABLE OR DISABLE (E/D)?
```

Enter E to enable the trace, or D to disable the trace. If the trace is being enabled,

```
160

OK
```

will be displayed. The "160" is a trace of the last line of the utility program. Now run the program you wish to test with line number tracing.

Note that the execution of any program (including utility programs such as this one) will cause the line number of the line currently being executed to be displayed, along with all other

information displayed by the program, while the trace is enabled. This should not adversely affect the operation of the program.

File name : TRACE

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 31	Track 26
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

NONE

```
1 PRINTCHR$(27);CHR$(28);
10 REM TRACE UTILITY UNDER OS-65D VERSION 3.0
20 REM
30 PRINT : PRINT "TRACE UTILITY" : PRINT
35 PRINT "WHEN BASIC'S TRACE FEATURE IS ENABLED, BASIC WILL PRINT"
37 PRINT "OUT EACH LINE NUMBER OF THE PROGRAM BEFORE IT IS EXECUTED"
38 PRINT
40 INPUT "ENABLE OR DISABLE (E/D)";A$
50 IF A$="E" THEN 100
60 IF A$="D" THEN 200
70 GOTO 40
100 REM
110 REM ENABLE
120 REM
130 REM THIS MUST ALL BE DONE AS ONE LINE!
140 REM
150 L=2011:POKEL,32:POKEL+1,216:POKEL+2,28:POKEL+3,234:POKEL+4,234
160 END
200 REM
210 REM DISABLE
220 REM
230 REM THIS MUST ALL BE DONE AS ONE LINE!
240 REM
250 L=2011:POKEL,24:POKEL+1,144:POKEL+2,2:POKEL+3,230:POKEL+4,200
260 END
```

ZERO

This utility program is used to zero the contents of a data file. This fills the entire data file with null (hex ØØ) characters which are ignored (skipped over) during BASIC input. You may find it advantageous to "zero" random data files before entering data into them in order to provide a "background" that is "transparent" (not seen) by a BASIC INPUT command. To zero a file, type:

```
RUN "ZERO"
```

The program output and the kind of input you may enter in response are as shown below. Any unacceptable response will result in an error message and/or a repeat of the request for input.

```
FILE ZERO UTILITY
COMPLETELY ERASES THE CONTENTS OF A DATA FILE
PASSWORD?
```

Enter PASS.

```
FILE NAME?
```

Enter the name of the file to be zeroed. The program continues with:

```
IS IT A NORMAL 12 (8 for a mini-floppy) PAGE DATA FILE?
```

Enter YES or NO. If NO is entered, the following message is output:

```
THEN HOW MANY PAGES PER TRACK?
```

Enter 1 through 12 (8 for a mini-floppy) to specify the number of 256 byte pages per track in the file.

The file will be zeroed and the program will terminate.

File name : ZERO

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 32	Track 27
Length :	2 Tracks	2 Tracks
Buffers for:		
Device #6:	YES	YES
Device #7:	NO	NO
Other :	18 BYTE OBJECT FILE	18 BYTE OBJECT FILE

Mini-floppy to 8" floppy program conversions

```
23Ø DATA 16Ø,Ø,162,12,152,153,126,58,2ØØ,2Ø8,25Ø

37Ø DISK!"CA 2E79=Ø8,1":GOSUB1ØØØØ

39Ø DISK!"CA 2E79=Ø8,2":GOSUB1ØØØØ

4Ø5 PG=12

41Ø ?:INPUT"IS IT A NORMAL 12 PAGE DATA FILE";AA$

44Ø IF PG<1 OR PG>12 THEN 43Ø
```

```
10 REM FILE ZEROING UTILITY OF OS-65D VERSION 3.0
20 REM
40 REM TO ZERO OUT DATA BUFFER: EXECUTE THIS;
50 REM
100 REM 2E7E A000    ZERO    LDY #0
110 REM 2E80 A208            LDX #8
120 REM 2E82 98              TYA
130 REM 2E83 997E32 ONE      STA $327E,Y  OR $3A7E
140 REM 2E86 C8              INY
150 REM 2E87 D0FA            BNE ONE
160 REM 2E89 EE852E          INC ONE+2
170 REM 2E8C CA              DEX
180 REM 2E8D D0F4            BNE ONE
190 REM 2E8F 60              RTS
200 REM
230 DATA 160,0,162,8,152,153,126,58,200,208,250
240 DATA 238,133,46,202,208,244,96
250 FORI=11902TO11919:READD:POKEI,D:NEXT
260 IFPEEK(8999)=50THENPOKE11909,50
280 POKE8955,126:POKE8956,46:X=USR(X):POKE8955,212:POKE8956,34
285 DEF FNA(X)=10*INT(X/16)+X-16*INT(X/16)
290 PRINT:PRINT"FILE ZERO UTILITY":PRINT
300 PRINT"COMPLETELY ERASES THE CONTENTS OF A DATA FILE"
310 PRINT
320 INPUT "PASSWORD";A$ : IF A$<>"PASS" THEN END
330 INPUT "FILE NAME";A$
340 IF LEN(A$)>6 THEN 330
350 IF LEN(A$)<6 THEN A$=A$+" " : GOTO 350
360 PN=11897
370 DISK!"CA 2E79=12,1":GOSUB 10000
380 IF FL<>0 THEN 405
390 DISK!"CA 2E79=12,2":GOSUB 10000
400 IF FL=0 THEN PRINT "** FILE NAME NOT IN DIRECTORY **":END
405 PG=8
410 PRINT:INPUT"IS IT A NORMAL 8 PAGE DATA FILE";AA$
420 IF MID$(AA$,1,1)="Y" THEN 450
430 INPUT "THEN HOW MANY PAGES PER TRACK";PG
440 IF PG<1 OR PG>8 THEN 430
450 REM
480 FOR I=T0 TO T9
490 T$=RIGHT$(STR$(I+100),2)
495 J$="3A7E":IFPEEK(8999)=50THENJ$="327E"
500 DISK!"SA "+T$+",1="+J$+"/"+RIGHT$(STR$(PG),1)
510 NEXT I
520 END
10000 REM
10010 REM FIND A$ IN DIRECTORY
10020 REM
10030 FOR I=PN TO PN+248 STEP 8
10040 B$=""
10050 FOR K=I TO I+5 : B$=B$+CHR$(PEEK(K)) : NEXT K
10060 IF A$<>B$ THEN 10090
10070 T0=FNA(PEEK(I+6)) : T9=FNA(PEEK(I+7))
10080 FL=1 : RETURN
10090 NEXT I
10100 FL=0 : RETURN
```

ASAMPL

Tutorial disk 5 contains a file named ASAMPL. This file holds a sample assembly language program which cannot be loaded while BASIC is in memory. If you are interested in assembly language programming and would like to examine this file, choose option 9 from the Tutorial Disk 5, then enter the command EXIT. The DOS prompt, A* will appear. Enter ASM to load the assembler/editor. When the '.' prompt appears, load ASAMPL by entering "!LO ASAMPL'. The '.' prompt will again appear. To list the program enter 'P'. Next, enter A3 to assemble and store the program. Finally run the program by entering '!GO 16ØØ'. The message

THIS IS A SAMPLE PROGRAM

should appear, then the system control is transferred to DOS as indicated by the A* prompt.

As indicated earlier, the loading of ASMAPL used memory which contained BASIC. In order to return to BASIC, then, BASIC must be reloaded by entering BA. When this is done, the message

```
OSI 9 DIGIT BASIC
COPYRIGHT 1977 BY MICROSOFT
96Ø1 BYTES FREE
```

will appear followed by the OK prompt.

File name : ASAMPL

File type : ASSEMBLER

	Mini-floppy specifications	8" floppy specifications
Location:	Track 34	Track 29
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

NONE

```
.A

   10                ;  SAMPLE ASSEMBLY LANGUAGE PROGRAM
   20                ;
   30                ;
   40                ;  EXTERNAL LABELS
   50                ;
   60 2D6A=          CRLF   = $2D6A
   70 2A51=          OS65D3 = $2A51
   80 2D73=          STROUT = $2D73
   90                ;
  100 1600                 * = $1600
  110               ;
  120 1600 206A2D   START  JSR CRLF
  130 1603 20732D          JSR STROUT
  140 1606 54              .BYTE 'THIS IS A SAMPLE PROGRAM',0
  140 1607 48
  140 1608 49
  140 1609 53
  140 160A 20
  140 160B 49
  140 160C 53
  140 160D 20
  140 160E 41
  140 160F 20
  140 1610 53
  140 1611 41
  140 1612 4D
  140 1613 50
  140 1614 4C
  140 1615 45
  140 1616 20
  140 1617 50
  140 1618 52
  140 1619 4F
  140 161A 47
  140 161B 52
  140 161C 41
  140 161D 4D
  140 161E 00
  150 161F 206A2D          JSR CRLF
  160 1622 4C512A          JMP OS65D3
  170               ;
  180                      .END
```

ATNENB

Tutorial disk 5 contains a program called ATNENB which is used to enable or disable BASIC's arctan function and the OS-65D V3.3 print extensions. Due to memory contraints in V3.3, only one of these features may be enabled at a time. To enable the features you desire, run the program ATNENB. You will be given an option of which feature you want. After making your selection, the system will configure itself to your needs. Use of the OS-65D V3.3 print extensions will be explained in Chapter 6.

File name : ATNENB

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 35	Track 3Ø
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	1 PAGE OBJECT FILE	1 PAGE OBJECT FILE

Mini-floppy to 8" floppy program conversions

NONE

```
10 PRINTCHR$(27);CHR$(21):REM   run under OS-65D V3.3 only!!!
20 :
30 PRINT"*** ATN  vs. PRINT  extensions ***":PRINT
40 PRINT"Only one may be enabled at a time!":PRINT
50 PRINT"Enter the number of your selection:"
60 PRINT:PRINT
70 PRINT"1> Enable ATN function
80 PRINT"2> Enable PRINT extensions"
90 PRINT:PRINT
490 :
500 INPUT"Your choice  (1 or 2) ";y$
510 IF (y$<>"1") AND (y$<>"2") THEN 500
520 y=VAL(y$):t=8278:f=14974:ctl=PEEK(2073):POKE 2073,96
525 :
530 IF y$="2" THEN FOR i=1 TO 9:READ a:NEXT:f=f+110
540 :
550 FOR i=0 TO 109:a=PEEK(f+i):POKE t+i,a:NEXT i
560 :
570 a=0558:READ b,c  :POKE a,b:POKE a+1,c
575 a=0828:READ b,c,d:POKE a,b:POKE a+1,c:POKE a+2,d
580 a=2642:READ b,c  :POKE a,b:POKE a+1,c
585 a=8643:READ b,c  :POKE a,b:POKE a+1,c
590 :
600 POKE 2073,ctl:a$="ATN function"
610 IFy$="2" THEN a$="PRINT extensions"
620 PRINT:PRINT:PRINTa$" enabled."
630 END
890 :
900 DATA 169,033,065,084,206,236,028,205,012
910 DATA 088,032,010,010,138,127,047,178,032
```

COLORS

Tutorial disk 5 contains a program called COLORS which may be used to aid in the adjustment of your color monitor. When the program COLORS is run, a test pattern is displayed on your video monitor. The pattern shows all sixteen colors that can be produced by your Challenger computer. You may then adjust your monitor using these colors as a guide. To exit the program, depress the SPACE BAR on the keyboard.

File name : COLORS

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Track 36	Track 31
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

NONE

```
10 REM ** COLOR SEQUENCE TEST PROGRAM **
20 POKE56832,7:PRINT!(28);
100 FORI=0TO15STEP2
300 READA$,B$:FORJ=1TO2
400 PRINT!(31,I)A$!(25)"      "!(31,I+1)B$!(25)
500 A$="                                ":B$=A$:NEXT
600 IFI<14THENPRINT
700 NEXT
1000 DATA"   0> YELLOW                  "
1010 DATA"  1> INVERTED YELLOW          "
1020 DATA"   2> RED                     "
1030 DATA"  3> INVERTED RED             "
1040 DATA"   4> GREEN                   "
1050 DATA"  5> INVERTED GREEN           "
1060 DATA"   6> OLIVE GREEN             "
1070 DATA"  7> INVERTED OLIVE GREEN     "
1080 DATA"   8> BLUE                    "
1090 DATA"  9> INVERTED BLUE            "
1100 DATA"  10> PURPLE                  "
1110 DATA" 11> INVERTED PURPLE          "
1120 DATA"  12> SKY BLUE                "
1130 DATA" 13> INVERTED SKY BLUE        "
1140 DATA"  14> BLACK                   "
1150 DATA" 15> INVERTED BLACK (WHITE)  "
20000 PRINT"             HIT THE SPACE BAR TO END THIS PROGRAM    ";
20010 DISK!"GO 2336":IFPEEK(9059)<>32THEN20010
20020 PRINT!(28);
20030 RUN"BEXEC*
```

MODEM

This is a BASIC program which will set up a machine code modem routine designed for use with a standard modem (with RS-232). The routine will operate with the modem ports on the Ohio Scientific C1P, C4P and C8P computers. The 63Ø and UTI board modem ports are exceptions to this and are not supported by this routine.

Under OSI 65D V3.3 (but not under V3.2 or ROM) input and output are directed through the Hazeltine 1420 emulator. Thus, you will appear as a Hazeltine terminal to the computer. Some 142Ø codes have been altered slightly to allow manipulation of OSI's color video. Others have been altered to facilitate use with OSI's Microsoft BASIC. Additionally, some codes have been added to support features the Hazeltine does not. In all cases, the changes to the Hazeltine command set have been done in a way to maximize software compatibility between the serial and video systems offered by Ohio Scientific.

There are two local commands:

CONTROL-D - Toggles the output back and forth between Full and Half duplex mode. (Sometimes echoed as a comma.)

CONTROL-B - Returns to BASIC if the routine is operating on a cassette system, or runs BEXEC* if it is operating on a disk system, effectively terminating the call.

Shift-0 is still used to output a delete character code. Since ROM BASIC doesn't process a backspace, the previous character will be omitted from the text but not on the video screen. The delete code will be displayed as a graphic backspace, a forward space and another graphic backspace on the ROM BASIC computers.

Suppose you want to call the local computer club computerized bulletin board, phone number (XXX-XXXX). You would,

1. Connect your modem to your computer via RS-232 (as discussed in your user's manual).
2. Type, RUN"MODEM"<RETURN>.
3. Dial up the bulletin board and connect the telephone to your local modem.
4. When you are through with modem operations, sign off using the remote's protocol and type <CTRL-B>.
5. Physically hang up the telephone to complete call termination.

File name : MODEM

File type : BASIC

	Mini-floppy specifications	8" floppy specifications
Location:	Tracks 37-38	Track 32
Length :	2 Tracks	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

NONE

```
10 REM MODEM PROGRAM
15 POKE133,80:CLEAR:FORI=1TO30:PRINT:NEXT
20 PRINT"MODEM ROUTINE LOADING"
30 Y=PEEK(2):Z=PEEK(64774)
40 IFZ=32THENGOSUB3000:GOTO60
50 GOSUB4000
60 FORI=1TO32:PRINT:NEXT:PRINT"MODEM READY"
70 X=USR(X):POKE63235,0:POKE64512,17
80 RESTORE:GOSUB500:IFY=4THENX=PEEK(8960):POKE133,X:RUN"BEXEC*"
90 END
500 PS=1:IFPEEK(9800)=32THENPS=2
510 IFY<>4ORZ<>32THENFORI=1TO48:READP:NEXT:RETURN
520 READP,C(1),C(2):IFPTHENPOKEP,C(PS):GOTO520
530 RETURN
540 DATA 9730,8,16
550 DATA 9743,7,15
560 DATA 9723,31,63
570 DATA 9736,31,63
580 DATA 9725,4,10
590 DATA 9738,29,59
610 DATA 9800,32,64
620 DATA 9636,101,75
630 DATA 9766,101,75
640 DATA 9770,101,75
650 DATA 9815,101,75
670 DATA 9670,125,123
680 DATA 9783,125,123
690 DATA 9682,95,164
990 DATA55296,0,1,0,0,0
1500 FORI=0+FTO216+F:READX
1510 IFX=-1THENX=INT(I/256)
1520 POKEI,X:NEXT
1530 RETURN
2000 DATA 32,13,37,173,0,240,74,144,6,173,1,240,32,67,35
2010 DATA 32,93,-1,240,239,201,2,240,22,201,4,240,21,72,32
2020 DATA 67,35,173,0,240,74,74,144,249,104,141,1,240,76,37
2030 DATA -1,76,13,37,173,63,-1,73,12,141,63,-1,208,225,138
2035 DATA 72,152,72
2040 DATA 169,1,32,190,252,32,198,252,208,5,10,208,245,240,83
2050 DATA 74,144,9,42,224,33,208,243,169,27,208,33,32,200,253
2060 DATA 152,141,19,2,10,10,10,56,237,19,2,141,19,2,168,138
2070 DATA 74,240,49,136,200,74,144,252,208,42,234,185,207,253,205
2080 DATA 21,2,208,38,206,20,2,240,43,160,5,162,200,202,208,253
2090 DATA 136,208,248,240,67,201,1,240,53,160,0,201,2,240,54,160
2100 DATA 192,201,32,240,48,169,0,141,22,2,141,21,2,169,2,141
2110 DATA 20,2,208,36,162,150,205,22,2,208,2,162,14,142,20,2
2120 DATA 141,22,2,169,1,32,190,252,32,207,252,74,144,3,76
2130 DATA 143,253,208,194,160,32,76,167,253,169,0,76,183,253
3000 GOSUB500
3005 IFY=4THENPOKE8955,34:POKE8956,82:F=21026:GOTO1500
3008 F=546:GOSUB1500
3010 POKE546,44:POKE592,96
3020 POKE559,251:POKE560,2:POKE576,251:POKE577,2
3030 POKE763,41:POKE764,127:POKE765,76:POKE766,45:POKE767,191
3040 POKE11,34:POKE12,2:RETURN
4000 GOSUB3000
4010 POKEF+65,141:POKEF+66,0:POKEF+67,223
4020 POKEF+68,174:POKEF+69,0:POKEF+70,223
4030 POKEF+193,141:POKEF+194,0:POKEF+195,223
```

```
4040 POKEF+196,173:POKEF+197,0:POKEF+198,223
4050 POKEF+1,68:POKEF+2,38
4060 POKEF+47,68:POKEF+48,38
4070 POKEF+5,252:POKEF+11,252:POKEF+34,252:POKEF+42,252
4080 IFY=4THENPOKE63235,52:POKE64512,2
4090 RETURN
```

COMPAR

Diskettes can be compared on a dual drive system as follows:

1. Place one disk in drive A and one in drive B.
2. Choose option 9, then enter

   EXIT

   The A* prompt will appear.
3. Enter

   CA Ø2ØØ=33,1 (for 8" floppies)

   or

   CA Ø2ØØ=39,1 (for 5" floppies)

   then enter

   CA 2ØØØ=39,2 (for 8" and 5" floppies)
4. Start running the program by entering

   GO Ø2ØØ
5. The disk COMPAR utility menu will appear on the screen. Select option 1.
6. Answer the sequence of questions so that the comparison will be made from drive A to drive B and over the range of tracks you wish to compare.
7. As each track is compared, its track number and sector specifications will appear on the screen.

   If any differences occur in a track, they will be listed with drive A on the left and drive B on the right, as follows:

```
Ø7
1
ØØØØ 7B 7C
ØØØ1 Ø5 Ø9
ØØ1Ø 96 87
Ø8
1
```

This shows three differences on track 7, sector 1. The first, second, and seventeenth bytes on this track/sector differ. The compare program found a 7B, Ø5, and 96 hex. on drive A and a 7C, Ø9, and 87 on drive B at these disk addresses.

File name : COMPAR

File type : OBJECT CODE

	Mini-floppy specifications	8" floppy specifications
Location:	Track 39	Track 33
Length :	1 Track	1 Track
Buffers for:		
Device #6:	NO	NO
Device #7:	NO	NO
Other :	NONE	NONE

Mini-floppy to 8" floppy program conversions

NOT TRANSFERABLE

Appendix 2 — DOS Command Summary

ASM	LOAD THE ASSEMBLER AND EXTENDED MONITOR. TRANSFER CONTROL TO THE ASSEMBLER.
BASIC	LOAD AND TRANSFER CONTROL TO BASIC.
CALL NNNN=TT,S	LOAD CONTENTS OF TRACK, "TT" SECTOR, "S" TO MEMORY LOCATION "NNNN".
D9	DISABLE ERROR 9. THIS IS REQUIRED TO READ SOME EARLIER VERSION FILES (V1.5, V2.Ø). (on 8" systems only)
DIR TT	PRINT SECTOR MAP DIRECTORY OF TRACK "TT". FOR EACH SECTOR, THE NUMBER OF PAGES IS GIVEN.
EM	LOAD THE ASSEMBLER AND EXTENDED MONITOR. TRANSFER CONTROL TO THE EXTENDED MONITOR.
EXAM NNNN=TT	EXAMINE TRACK. LOAD ENTIRE TRACK CONTENTS, INCLUDING FORMATTING INFORMATION, INTO LOCATION "NNNN".
GO NNNN	TRANSFER CONTROL (GO) TO LOCATION "NNNN".
HOME	RESET TRACK COUNT TO ZERO AND HOME THE CURRENT DRIVE'S HEAD TO TRACK ZERO.
INIT	INITIALIZE THE ENTIRE DISK. I.E. ERASE THE ENTIRE DISKETTE (EXCEPT TRACK Ø) AND WRITE NEW FORMATTING INFORMATION ON EACH TRACK.
INIT TT	SAME AS "INIT", BUT ONLY OPERATES ON TRACK "TT".
IO NN,MM	CHANGES THE INPUT I/O DISTRIBUTOR FLAG TO "NN", AND THE OUTPUT FLAG TO "MM".
IO ,MM	CHANGES ONLY THE OUTPUT FLAG. (See page 54)
IO NN	CHANGES ONLY THE INPUT FLAG. (See page 54)
LOAD FILNAM	LOADS NAMED SOURCE FILE, "FILNAM" INTO MEMORY.
LOAD TT	LOADS SOURCE FILE INTO MEMORY GIVEN STARTING TRACK NUMBER "TT".
MEM NNNN,MMMM	SETS THE MEMORY I/O DEVICE INPUT POINTER TO "NNNN", AND THE OUTPUT POINTER TO "MMMM".

PUT FILNAM	SAVES SOURCE FILE IN MEMORY ON THE NAMED DISK FILED "FILNAM".
PUT TT	SAVES SOURCE FILE IN MEMORY ON TRACK "TT" AND FOLLOWING TRACKS.
RET ASM	RESTART THE ASSEMBLER.
RET BAS	RESTART BASIC.
RET EM	RESTART THE EXTENDED MONITOR.
RET MON	RESTART THE FROM MONITOR (VIA RST VECTOR).
SAVE TT,S=NNNN/P	SAVE MEMORY FROM LOCATION "NNNN" ON TRACK "TT" SECTOR "S" FOR "P" PAGES.
SELECT X	SELECT DISK DRIVE "X" WHERE "X" CAN BE; A, B, C, OR D. SELECT ENABLES THE REQUESTED DRIVE AND HOMES THE HEAD TO TRACK Ø.
XQT FILNAM	LOAD THE FILE, "FILNAM" AS IF IT WAS AN OBJECT FILE, AND TRANSFER CONTROL TO LOCATION $3A7E (317E on 8" V3.2; 327E on 5" V3.2)
XQT TT	LOAD THE FILE BEGINNING ON TRACK "TT" AS IF IT WAS AN OBJECT FILE AND TRANSFER CONTROL TO LOCATION $3A7E (317E on 8" V3.2; 327E on 5" V3.2)

NOTES:

- ONLY THE FIRST 2 CHARACTERS ARE USED IN RECOGNIZING A COMMAND. THE REST UP TO THE BLANK ARE IGNORED.

- THE LINE INPUT BUFFER CAN ONLY HOLD 18 CHARACTERS INCLUDING THE RETURN.

- THE COMMAND LOOP CAN BE REENTERED AT $2A51.

- FILE NAMES MUST START WITH A "A" TO "Z" AND CAN BE ONLY 6 CHARACTERS LONG.

- THE DIRECTORY IS ALWAYS MAINTAINED ON DISK. THIS PERMITS THE INTERCHANGE OF DISKETTES.

- THE FOLLOWING CONTROL KEYS ARE VALID:

CONTROL - Q	CONTINUE OUTPUT FROM A CONTROL-S.
CONTROL - S	STOP OUTPUT TO THE CONSOLE.

- COMMANDS CAN BE USED IN THE BASIC MODE IN THE FORM DISK!"DOS" WHERE DOS REPRESENTS ONE OF THE COMMANDS ABOVE.

Appendix 3 BASIC Command Summary

The entries are organized alphabetically according to Keywords used. Each entry consists of the general syntax, examples where appropriate, and a brief description.

The following notation is used:

[n]	see page n of the OS-65D Tutorial and Reference Manual (this manual)
(*)	cannot be used in the immediate (direct) mode; must be used with a program statement number.
(**)	can only be used in the immediate (direct) mode; must not be used within a program.
(2)	not available under OS-65D V3.3.
(3)	available only under OS-65D V3.3.
{n}	see page n of the OSI BASIC Reference Manual
ae	a numeric constant or arithmetic expression (see {3})
re	a logical constant or relational expression (see {4})
se	a string constant or expression (see {4})
dos	a 65D Disk Operating System (DOS) command.
e	a constant or expression.
V	a variable
c	a constant
nv	a numeric variable
iv	an integer variable
sv	a string variable
niv	a nv or iv
rae	a re or ae
FILE	a disk file name
loc	a memory location address
sn	a program statement number
dev	an OS-65D device number (see [54])

ABS — ABS(ae)

A function. Returns the absolute value of its argument. {19}

AND — re AND re

```
IF X<15 AND X>=Ø THEN 1ØØ
```

A bitwise Boolean AND operator. re AND re will be TRUE only when both of the operands have the value TRUE. {4}

ASC — ASC(se)

```
ASC(X$)      ASC("BIG")
```

A function. Returns the ASCII value in decimal of the first character in the argument {20}

ATN — ATN(ae) (-1<ae< 1)

```
ATN(Ø.431)
```

A function. Returns the arctangent of the argument {20} (2) [188]

CHR$

CHR$(ae) (Ø≤ae≤ 255)
CHR$(66)
A function. Returns the character whose decimal ASCII value is the greatest integer less than or equal to the argument. {21}.

CLEAR

CLEAR
Clears the program variable table and restores the data pointer (*) {17}

CLOSE

DISK CLOSE, dev (dev =6 or dev =7)
Closes a disk file that has been previously opened. {28} [15]

CONT

CONT
Restarts as program whose execution has been interrupted by a STOP or END statement or a CTRL-C. {15} (**)

COS

COS(ae)
A function. Returns the cosine of the argument. {20}

DATA

DATA c, c, c, ...
DATA 1.7, "BIG", 173, -812
Establishes a list of constants to be input by the program via the READ statement {6}

DEF FN

DEF FNnv(nv) = ae
DEF FNA(X) = X*7+3
Defines a single variable function for future use within the program segment {23} (*)

DIM

DIM v(ae, ae,...),...
DIM A(20), B$(6,7)
Declares the variables specified to be subscripted. {18},

DISK!

DISK! "dos"
DISK! "IO 5,6"
DISK! "LOAD FILE"
Permits 65D DOS commands to be used within a BASIC program. [2Ø2]

DISK CLOSE

see CLOSE

DISK FIND

see FIND

DISK GET

see GET

DISK OPEN

see open

DISK PUT

see PUT

EDIT

EDIT sn
EDIT 1ØØ
Returns line sn for editing. The short form is ! sn. (**) [71] (3)

END

END
Terminates program execution {13}

EXIT

EXIT
Transfers control to the DOS mode {28} [53] (**)

EXP

EXP(ae) ae< 88.Ø29619
  EXP(41.662)
A function. Returns e = 2.71828...raised to the power equal to the value of the argument. {19}

FIND

DISK FIND, se
  DISK FIND, "BIG"
Beginning at current file pointer location, the data file is searched for the specified string, the pointer is set to the end of the field in which it is found. An unsuccessful search results in a #D error. [96] (3)

FN

see DEF FN

FOR

FOR niv = ae TO ae
FOR niv = ae TO ae STEP ae
   FOR X% = 15 TO 45 STEP 5
Opens program loop. End of the loop is indicated by the statement NEXT or NEXT niv. STEP is used to define an increment other than 1 for niv for each iteration of the loop. In the example, the loop is executed 7 times {12} (*)

FRE

FRE(X) X is a dummy variable
A function. Returns the number of bytes of memory in the workspace that are unused. Save the program before using FRE. {17}

GET

DISK GET, niv
  DISK GET, 15
Brings the record number niv from the disk to buffer #6 and sets the I/O pointers to the beginning of the record {28} [17]

GOSUB

GOSUB sn
  GOSUB 1ØØØ
Program control is transferred to statement number sn. When the statement RETURN is encountered, control goes back to the statement following sn {23}

GOTO

GOTO sn
  GOTO 1ØØØ
Program control is transferred to statement number sn. {11}

IF

IF rae GOTO sn
IF rae THEN sn
If the value of rae is TRUE (arithmetic expressions are considered to be TRUE if they have a value other than Ø) program control is transferred to statement sn.
IF rae THEN S (S is a program statement)
If the value of rae is TRUE, statement S is executed {11}

INPUT

INPUT V, V, ...
  INPUT X, Y, A$
Prompts for keyboard input to the specified variables {6} (*)

INPUT#

INPUT#dev, V, V, ...
  INPUT#6, A, B, Q$
Input is from device number dev to the specified variables. {9} [13] (*)

INT

INT (ae)
  INT (-16.8)
A function. Returns the greatest integer less than or equal to the argument {19}

LEFT$

LEFT$(se, ae) ae >Ø
  LEFT$("ABCDE", 3)
A function. Truncates ae to an integer and returns that leftmost number of characters from string se. In the example, "ABC" is returned. {21}

LEN

LEN(se)
  LEN(A$)
A function. Returns the length of the string se {21}

LET

LET V = e
  LET A$ = "BIG"
Assignment statement. Keyword LET is optional. {6}

LIST

LIST
LIST sn-sn
    LIST 1ØØ-2ØØ
    LIST - 1ØØØ
    LIST 2ØØ-
Lists the program in the workspace between the two specified statement numbers. If the first (second) statement number is omitted, the default is the beginning (end) of the program. {15}

LIST#

LIST#dev
  LIST#4
Same as LIST, but the listing is sent to device number dev. {9, 15}

LOG

LOG(ae) ae>Ø
  LOG14.8
A function. Returns the natural logarithm (log to the base e) of the argument. {19}

MID$

MID$(se, ae, ae) first ae>Ø, second ae≥Ø
  MID$("ABCDEFG", 2, 3)
A function. In the example, A string of length 3 starting at position 2 is returned; i.e. "BCD". If the second ae is omitted, the sting returned goes to the end of se. {21}

NEW

NEW
Clears the workspace to prepare for creation of a new program {15}

NEXT

see FOR

NOT

NOT re
  NOT (A>5)
A bitwise Boolean NOT operator. Reverses the truth value of the operand re. {3}

NULL

NULL iv Ø≤iv≤ 8
Inserts iv zeros at the beginning of each line as it is stored on tape. {27} {2}

ON

ON ae GOTO sn, sn,...
ON ae GOSUB sn, sn,...
  ON a GOTO 1ØØ, 2ØØ
Depending upon the value of ae (truncated to an integer) program control passes to the ae-th statement in the list of statement numbers {12, 24}

OPEN

DISK OPEN, dev, "FILE" (dev = 6 or 7)
Opens the disk file FILE for sequential (dev=6 or 7) or random access (dev=6 only) {28} [15]

OR

re OR re
  IF A >5 OR A <2 THEN 1ØØ
A bitwise Boolean OR operator. re OR re is FALSE only when both of the operands are FALSE. {3}

PEEK

PEEK(loc)
A function. Returns the value stored in memory location loc {25}

POKE

POKE loc, ae ae is an integer.
  POKE 11686, 17
The value ae is stored in memory location loc {25}

POS

POS(X) X is a dummy variable.
A function. In or following a PRINT statement, returns the current position (between Ø and 132) of the cursor {9}

PRINT

PRINT e, e,...
  PRINT A, B$; C$
Outputs the values stored in the list of expressions. The keyword PRINT can be replaced by a question mark. {7}

PRINT#

PRINT#dev, e, e,...
Same as PRINT, but output is directed to device number dev instead of the screen. {7} [13]

PRINT!

PRINT!(HOC), e, e,... (HOC=Hazeltine Operation code-see [223])
PRINT!(28) X$, A, B, C
Depending on the value of HOC, certain screen characteristics and cursor positons are selected before beginning output of expression values; emulates certain Hazeltine terminal capabilities. [223] (3)

PRINT CHR$

see CHR$

PRINT&

PRINT&(X, Y), e, e,...
PRINT&(10, 20) A, B$
Moves the screen cursor to screen position (X, Y) (Ø, Ø) = upper left corner) before beginning output of expression values. Identical to:
PRINT!(17,X,Y), e, e,... [79] (3)

PRINT USING

PRINT USING se ae, ae, ...
  PRINT USING "####.##" 6.87304
Used to format numeric output; se must be a string expression made up of a decimal point and/or #'s. In the example the output format specified results in printing 6.87 (with three leading blanks) [73] (3)

PUT

DISK PUT
Follows a previous DISK GET; places the current record back to the disk. [28} [17]

READ

READ V, V, V,...
  READ A, B$, C
Inputs constants that are specified by DATA statements in the same program into the specified variables {6} (*)

REM

REM any remark
  REM THIS IS A TEST PROGRAM
Used for program documentation. Everything appearing after REM is ignored on execution of that line {16} (*)

RESTORE

RESTORE
Resets the pointer in a program's DATA list to the first item. {7} (*)

RETURN

See GOSUB

RIGHT$

RIGHT$(se, ae) ae≥Ø
RIGHT$("ABCDEF",2)
A function. Truncates ae to an integer and returns that number of rightmost characters. In the example, "EF" is returned. {21}

RND

RND(ae)
RND(-16)
A function. Returns a number between Ø and 1. Can be used repeatedly to generate a sequence of pseudo-random values. If ae>Ø, the argument is a dummy argument. If ae=Ø, RND returns the previous value again. If ae<Ø, ae functions as a "seed" and RND starts a new sequence. The sequence repeats after a certain period determined by the seed. {19}

RUN

RUN
Starts execution of the program in the workspace at the first statement.
RUN sn
Starts execution of the program in the workspace at statement number sn.
RUN "FILE"
Leads the program from disk file
FILE and starts execution.
RUN "TT" (TT = a disk track number)
Loads the program from the disk file beginning at track TT and starts execution. {15}

SGN

SGN(ae)
A function. Returns +1 if ae >Ø, Ø if ae = Ø, -1 if ae <Ø. {19}

SIN

SIN(ae)
A function. Returns the value of the sine of the argument ae. {20}

SPC

SPC(ae)
PRINT "A"; SPC(5); "B"
A function. Used to print ae spaces in a PRINT sequence {9}

SQR

SQR(ae) ae≥Ø
A function. Returns the square root of the argument ae. {20}

STEP

See FOR

STOP

STOP
Halts execution of a program and prints a BREAK message indicating the statement number of the STOP statement {13}

STR$

STR$(ae)
STR$(6.71)
A function. Returns the value of the argument ae as a string. {21}

TAB

TAB(ae) ae is an integer
TAB(1Ø)
A function. Used in a PRINT statement to move the print position for the next character to position ae+1 on the print line. {8}

TAN

TAN(ae)
A function. Returns the tangent of the argument. {20}

THEN

See IF

TO

See FOR

TRAP

TRAP sn
If an error is encountered in a program after this statement, then control transfers to statement sn. TRAP Ø disables error trapping. [71] (3)

USR

USR(ae)
Y = USR(X)
Transfers control to a machine language routine at a location determined previously by appropriate POKES. ae may be an input parameter (and USR(ae) an output parameter) or ae may be a dummy parameter. {34}

VAL

VAL(se)
VAL("6.31")
A function. It is the opposite of STR$; returns the numeric value of the string expression se if se represents a number. Otherwise, Ø is returned.

WAIT

WAIT loc, J 0 ≤J ≤255
Halts program execution, Reads the contents of location loc and AND's the result (bitwise) until a nonzero result is obtained, then resumes program execution.
WAIT loc, J, K 0 ≤J, K≤ 255
Halts program execution, reads the contents of location loc, exclusive OR's that value (bitwise) with K, and then AND's the result with J until a nonzero result is obtained ; then resumes execution {25} {2}

The syntax for editing a line is as follows:

Ø<LN<64ØØØ <CR> = carriage return or RETURN

EDIT LN<CR> or !LN<CR> - Edit the statement with the line number LN.

EDIT!<CR> or !!<CR> - Edit the same line that was just

EDIT<CR> or !<CR> - Edit the line immediately following the line that was just edited.

The line with its line number will be displayed following the <CR>. If the line number LN does not exist, the statement with the next line number will be displayed (e.g., typing EDITØ or !Ø will always give the first line of the program). After the statement is displayed, the cursor will reside at the end of that line. The following commands are used for the actual line editing.

CTRL-P - non-destructive forward space. Moves the cursor one space to the right.

CTRL-H - non-destructive backspace. Moves the cursor one space to the left.

RUBOUT or SHIFT-O - single character delete. The editor makes the correct delete keys operational as well as the old ones (i.e., the RUBOUT key as well as SHIFT-O will work on the OSI polled keyboard when the editor is enabled).

SHIFT-P - entry delete. This will erase the line currently being edited, leaving the line in the text as it was before it was edited.

CTRL-R - non-destructively moves the cursor to the "rear" of the statement.

CTRL-F - non-destructively moves the cursor to the "front" of the statement.

CTRL-I - non-destructively moves the cursor forward to the next Tab Position (positions 1, 8, 15, 22, 29, 36, 43, 50, 57, 64, 71).

CTRL-T - retypes the statement you are currently editing.

<CR> or RETURN - enters the line as written or viewed. The line will look (to the BASIC interpreter) as if it was typed in by the user from scratch.

Character insertions and deletions can be accomplished anywhere by using the commands for non-destructive movement of the cursor. After the cursor is positioned, the user can type in insertions or delete unwanted characters. NOTE: 1) Characters are inserted to the left of the character on which the cursor resided, 2) the character on which the cursor resides is deleted until the end of the line is reached. The characters to the left will be deleted if the cursor resides at the end of a line.

Appendix 5 — Error Message Codes

A. DOS Error Message Codes

1 - Can't Read Sector (Parity Error).

2 - Can't Write Sector (Reread Error).

3 - Track Zero is Write Protected Against that Operation.

4 - Diskette is Write Protected.

5 - Seek Error (Track Header Doesn't Match Track).

6 - Drive Not Ready.

7 - Syntax Error in Command Line.

8 - Bad Track Number.

9 - Can't Find Track Header Within One Rev of Diskette.

A - Can't Find Sector Before One Requested.

B - Bad Sector Length Value.

C - Can't Find that Name in Directory.

D - Read/Write Attempted Past End of Named File.

B. BASIC Error Message Codes

BS	Bad subscript: Matrix outside DIM statement range, etc.
CN	Continue Errors: Attempt to inappropriately continue from BREAK or STOP.
DD	Double Dimension: Variable dimensioned twice. Remember subscripted variables default to dimension 1Ø.
FC	Function Call Error: Parameter passed to function out of range.
ID	Illegal Direct: INPUT and DEF statements cannot be used in direct mode.
LS	Long String: String longer than 255 characters.

NF	NEXT without FOR.
OD	Out of Data: More reads than data.
OM	Out of Memory: Program too big or too many GOSUBs, FOR-NEXT loops or variables.
OV	Overflow: Result of calculation too large.
RG	RETURN without GOSUB.
SN	Syntax Error: Typo, etc.
ST	String Temporaries: String expression too complex.
TM	Type Mismatch: String variable mismatched to numeric variable.
UF	Undefined Function.
US	Undefined Statement: Attempt to jump to nonexistent line number.
/Ø	Division by Zero.
OS	Out of String Space: Same as OM.

Appendix 6

POKE AND PEEK LIST

As systems develop, different locations are committed to hold parameters. Many of these parameters have been mentioned in the text material. These parameters are collected here, along with some other useful parameters which may be needed by an advanced programmer. Users of the video systems and systems that include certain options and accessories (e.g., Home Security, Remote Control, High Resolution Graphics, etc.) may need to POKE or PEEK other parameter locations. These locations are fully documented in the appropriate User's Manuals.

Caution: care must be taken when POKEing any of these locations to avoid system errors and subsequent software losses.

LOCATION DECIMAL	LOCATION HEX	NORMAL CONTENTS (DEC)	COMMENTS
23	17	132	Terminal width (number of printer characters per line). The default value is 132. Note, this is not to be confused with the video display width (64 characters).
24	18	112	Number of characters in BASIC's 14-character fields (112 characters = 8 fields) when outputting variables separated by commas.
12Ø-121	78-79	127 5Ø	Lo-Hi byte address of the beginning of BASIC work space (note 127 = $7F, 5Ø = $32). Normal contents of Location 121 is 58 on V.3.3 and 49 on Serial Systems.
741	2E5	1Ø	Control location for "LIST." Enable with a 76, disable with a 1Ø.
750	2EE	1Ø	Control location for "NEW." Enable with a 78, disable with a 1Ø.
1797	7Ø5	32	Controls line number listing of BASIC programs, enable with a 32, disable with a 44.
2Ø73	819	173	"CONTROL C" termination of BASIC programs. Enable with 173, disable with 96.

LOCATION DECIMAL	HEX	NORMAL CONTENTS (DEC)	COMMENTS
22ØØ	898	-	The monitor ROM directs Tract Ø to load here at $22ØØ.
2888	B48	27	A 27 present here allows any null input (carriage return only) to force into immediate jumping out of the program. Disable this with a Ø. Location 8722 must also be set to Ø.
2893	B4D	55	Alternate "break on null input" enable/disable location.
2894	B4E	Ø8	A null input will produce a "REDO FROM START" message when 2893 and 2894 are POKEd with 28 and 11 respectively.
2972	B9C	58	Normally a comma is a string input termination. This may be disabled with a 13 (see 2976).
2976	BAØ	44	A colon is also a string input terminator, this is disabled with a 13 (see 2976).
87Ø8	22Ø4	41	Output flag for peripheral devices.
8722	2212	27	Null input if = ØØ, normal input if = 27.
89Ø2	22C6	ØØ	Determines which registers (less 1) RTMON scans.
8917	22D5	-	USR (X) Disk Operation Code: Ø-write to Drive A 3-read from Drive A 6-write to Drive B 9-read from Drive B
8954	22FA	-	Location of JSR to a USR function. Present to JSR $22D4, i.e., set up for USR (X) Disk Operation.
896Ø	23ØØ	-	Has page number of highest RAM location found on OS-65D's cold start boot in. This is the default high memory address for the assembler and BASIC.
8993	2321	-	I/O Distributor INPUT flag (see p. 54)
8994	2322	-	1/O Distributor OUTPUT flag(see p. 54)

LOCATION DECIMAL	HEX	NORMAL CONTENTS (DEC)	COMMENTS
8995	2323	-	Index to current ACIA on 55Ø board. If numbered from 1 to 15 the value POKEd here is a 2 times the ACIA number.
8996	2324	-	Location of a random number seed. This location is constantly incremented during keyboard polling.

(Note: Locations 8998 through 9005, 9132-9133, and 9155-9156 are used for Disk Buffer #6 (I/O Flag Bit 5 device) usage parameters.)

8998-8999	2326-2327	126 *	LO-HI byte address for the start of Buffer #6 (*contents vary: 58 on all V3.3; 50 on 5" V3.2; 49 on 8" V3.2)
9ØØØ-9ØØ1	2328-2329	126 *	LO-HI byte address for the end of Buffer #6 (*contents vary: 66 for 5" V3.3; 7Ø for 8" V3.3; 58 for 5" V3.2; 61 for 8" V3.2)
9ØØ2	232A	-	First track of Buffer #6 File
9ØØ3	232B	-	Last track of Buffer #6 File
9ØØ4	232C	-	Current track in Buffer #6
9ØØ5	232D	-	Buffer #6 Dirty Flag (if contents is non-zero, then data has been written to the buffer, but has not yet been transferred to the disk)

(Note: Locations 9ØØ6 through 9Ø13, 9213-9214, 9238-9239 are used for Disk Buffer #7 (I/O Flag Bit 6 device) usage parameters)

9ØØ6-9ØØ7	232E-232F	126 *	LO-HI Byte address for the start of Buffer #7 (*contents vary: 58 on 5" 3.2; 61 on 8" V3.2; 66 on 5" V3.3; 7Ø on 8" V3.3)
9ØØ8-9ØØ9	233Ø-2331	126 *	LO-HI Byte address for the end of Buffer #7 (*contents vary: 66 on 5" V3.2; 73 on 8" V3.2; 74 on 5" V3.3; 82 on 8" V3.3)

LOCATION DECIMAL	HEX	NORMAL CONTENTS (DEC)	COMMENTS
9Ø1Ø	2332	-	First track of Buffer #7 File
9Ø11	2333	-	Last track of Buffer #7 File
9Ø12	2334	-	Current track in Buffer #7
9Ø13	2335	-	Buffer #7 Dirty Flag (Ø = Clean; see comment for location 9ØØ5)
9Ø98- 9Ø99	238A- 238B	- -	Pointer to Memory Storage Input (Lo and Hi Byte). Memory is dedicated for use as file.
91Ø5- 9106	2391- 2392	- -	Pointer to Memory Storage Output (Lo and Hi Byte). Use of memory as a file.
9132- 9133	23AC- 23AD	126 *	LO-HI Byte address of Buffer #6 current input. (* 5Ø on 5" V3.2; 49 on all other systems)
9155- 9156	23C3- 23C4	126 *	LO-HI Byte address of Buffer #6 current output. (*5Ø on 5" V3.2; 49 on all other systems)
9213- 9214	23FD- 23FE	126 *	LO-HI Byte address of Buffer #7 current input. (*62 on 5" V3.2; 61 on all other systems)
9238- 9239	2416- 2417	126 *	LO-HI Byte address of Buffer #7 current output. (*62 on 5" V3.2; 61 on all other systems)
9368	2498	-	Indirect File Input Address (Hi Byte) (Lo = ØØ) (For use, see BASIC Reference Manual, Chapter 12)
9554	2552	-	Pointer to Indirect File (Hi Byte only) for output (Lo = ØØ)
9666	25C2	Ø	When POKEd with N (Ø-63) and a LIST command is given, this will move the left hand margin to the right N spaces (dashes will echo on the left unless the cursor is removed).
9667	25C3	215	When POKEd with N (2Ø7-215) and a LIST command is given, this will move the scroll up 4*(215-N) lines.
968Ø	23DØ	95	Cursor symbol character designation, for video screen.

LOCATION DECIMAL	LOCATION HEX	NORMAL CONTENTS (DEC)	COMMENTS
9682-9683	25D2-25D3	- -	Next Position for Cursor on video screen (HI and LO Bytes)
977Ø	262A	64	Display control parameters. Single Space = 64; Double Space = 128; Quad Space = 255; Two columns = 32.
9796	2644	-	Entry point to Keyboard Swap Routine
9822	265D	-	Sector for USR(X) on Disk.
9823	265F	-	Page Count for USR(X). Read or Write.
9824	266Ø	-	Pointer to memory for USR(X). (Lo and Hi Bytes) USR(X) will reside in location pointed to.
9826	2662	-	Contains track number for USR(X) on disk
9976	26F8	-	Disable ":" Terminator. See Location 2976 comments.
1Ø95Ø	2AC6	*	Console terminal number. (*1 on Serial Systems; 2 on Video Systems)
11511	2CF7	-	Used by Disk Page Ø/1 Swap Used by Random Access File
12Ø42	2FØA	-	Sets Number of records per track for data file use (see chapter 4 or 6)
12076	2F2C	-	Sets record length for data file use (see chapter 4 or 6)
13026	32E2	171	Selects cursor character (V3.3 only)
13743	35AF	32	Selects Flashing cursor; 44 selects non-flashing cursor. (V3.3 only)

Appendix 7

ASCII Character Codes

CODE	CHAR	CODE	CHAR	CODE	CHAR
ØØ	NUL	2B	+	56	V
Ø1	SOH	2C	,	57	W
Ø2	STX	2D	–	58	X
Ø3	ETX	2E	.	59	Y
Ø4	EOT	2F	/	5A	Z
Ø5	ENQ	3Ø	Ø	5B	[
Ø6	ACK	31	1	5C	/
Ø7	BEL	32	2	5D	]
Ø8	BS	33	3	5E	^
Ø9	HT	34	4	5F	_
ØA	LF	35	5	6Ø	
ØB	VT	36	6	61	a
ØC	FF	37	7	62	b
ØD	CR	38	8	63	c
ØE	SO	39	9	64	d
ØF	SI	3A	:	65	e
1Ø	DLE	3B	;	66	f
11	DC1	3C	<	67	g
12	DC2	3D	=	68	h
13	DC3	3E	>	69	i
14	DC4	3F	?	6A	j
15	NAK	40	@	6B	k
16	SYN	41	A	6C	l
17	ETB	42	B	6D	m
18	CAN	43	C	6E	n
19	EM	44	D	6F	o
1A	SUB	45	E	7Ø	p
1B	ESC	46	F	71	q
1C	FS	47	G	72	r
1D	GS	48	H	73	s
1E	RS	49	I	74	t
1F	US	4A	J	75	u
2Ø	SP	4B	K	76	v
21	!	4C	L	77	w
22	"	4D	M	78	x
23	#	4E	N	79	y
24	$	4F	Ø	7A	z
25	%	50	P	7B	{
26	&	51	Q	7C	}
27	'	52	R	7D	~
28	(	53	S	7E	÷
29	)	54	T	7F	DEL
2A	*	55	U		

Appendix 8

V3.3 PRINT Command Summary

A. Arranged According to Function

(These commands must be used in PRINT statements)

Display Size

!(2Ø)	Selects "wide letter" display (32 x 32 on C4P and C8P, 12 x 24 on C1P, clears screen and homes cursor to upper left screen corner.
!(21)	Selects "narrow letter" display (32 x 64 on C4P and C8P, 24 x 48 on C1P), clears screen, and homes cursor to upper left screen corner.
!(22, w, h)	Selects print window w characters wide and h characters high. Upper left window corner is at current cursor position; screen is not cleared.

Cursor Control

Single Step

CHR$(8)	Back one space.
CHR$(16)	Forward one space.
!(12)	Up one space.
!(11)	Down on space.
CHR$(1Ø)	Down one space.

Multistep

CHR$(13)	Back to front of line.
CHR$(14)	Forward to next eight space tab set (seven space for left-most field).

Anywhere

!(17, x, y)	Relocate to x, y(Ø, Ø at upper left corner).
&(x, y)	Relocate to x, y (Ø, Ø at upper left corner).

Home

!(18)	Relocate to Ø, Ø (upper left corner).

Insert

!(26)	Inserts line at cursor position; lower lines scroll down.

Clear

Line

!(15)	Clears from cursor to end of line.
!(19)	Clears entire line (lower lines move up).

Screen

!(24)	Clears from cursor to end (lower right) of window.
!(28)	Clears entire screen and homes cursor in window.

Color

Color Select

!(1)	Selects color Ø as cell background.
!(25)	Selects normal black/white display mode (i.e., black background, white character).
!(31, n)	Selects color n as cell background.

Color Change

!(2, n, m)	Changes all displayed cells of background color m to background color n.
!(29, n)	Clears all displayed cells of background color n (i.e., cell background is changed to black and character is replaced with a blank).

Cursor Sensing

!(5)	Sends information for current cursor position x, y, to string variable in following INPUT statement. Information is in the form of two characters for which (x + 65) is the ASCII code. Line feed follows the INPUT statement used with !(5).
!(33)	Sends character at cursor position to string variable in following INPUT statement. Line feed follows the INPUT statement used with !(33).

B. Arranged According to ASCII Codes.

!(1)	Color Ø select
!(2, n, m)	Color change
!(5)	Cursor position sensing
CHR$(8)	Cursor backspace
CHR$(1Ø)	Cursor down one space
!(11)	Cursor down one space
!(12)	Cursor up one space
CHR$(13)	Cursor tab forward
CHR$(14)	Carriage return
!(15)	Partial line clear
CHR$(16)	Cursor forward one space
!(17, x, y)	Cursor relocation to position (x, y)
!(18)	Cursor home
!(19)	Total line clear (lower lines up)
!(2Ø)	Wide character select
!(21)	Narrow character select
!(22, w, h)	Window select
!(24)	Partial screen (or window) clear
!(25)	Color black/white select
!(26)	Line insert (lower lines down)
!(28)	Total screen clear (cursor home)
!(29, n)	Selective color clear
!(31, n)	Color n select
!(33)	Character pick-up

Appendix 9 Extended Utilities Command Summary

RESEQUENCER

System must first be booted under V3.2. Enable by the command RUN"RSEQ and then type "E". Then LOAD the program to be resequenced into the workspace.

The syntax for the RSEQ command is as follows:

NLN = new line number Ø<=NLN<64ØØØ

OLN = old line number Ø<=OLN<64ØØØ

INC = increment between line number Ø<INC<256

RSEQ<CR>	- resequence starting with the line number 1Ø at the first line and renumber the lines in increments of ten.
RSEQ NLN<CR>	- resequence using NLN as the first line number and renumber the lines that follow by increments of ten.
RSEQ NLN,OLN<CR>	- resequence starting at line OLN with line number NLN and renumber the lines that follow by increments of ten.
RSEQ NLN,OLN,INC<CR>	- resequence starting at line OLN with the line number NLN and renumber the lines that follow by increments of INC.
RSEQ NLN,,INC<CR>	- resequence starting with the first line as NLN and renumber the lines that follow by increments of INC.
RSEQ ,,INC<CR>	- resequence starting with the first line and renumber the lines that follow by increments of INC.

REPACKER

System must first be booted under V3.2. Simply RUN"REPACK, insert the disk which contains the program to be repacked and enter the program's file name. Select the method of repacking (remove blanks, REM's or both). When the operations are completed, a message will appear reporting the number of bytes removed from the program.

BUFFER CREATOR

System must first be booted under V3.2. Enable with the command RUN"BUFFER and then type "E". Then LOAD the program you wish to work with (the source) and use the BYTE command according to the following syntax:

Ø=<NB<64ØØØ

BYTE NB<CR>- moves source leaving NB free bytes in between the operating system and the source.

BYTE<CR> - reports the start of the buffer, the starting byte of your program and the number of bytes between the source and the operating system.

DATA FILE COPIER

System must first be booted under V3.2. To use, type RUN"DATRAN and respond to the prompts for input file name, output file name and location (device A, B, C, D) for each file. The input file is then copied onto the output file.

BASIC DISASSEMBLER

Can be run with the system booted under any version of 65D. To use, type RUN"DISASM and then enter the addresses in decimal for the beginning and end of the disassembly.

GENERAL STRING ORIENTED SORT

Can be run with the system booted under any version of 65D. To use, type RUN"GSOSRT and respond to the prompts for type of file and type of sorting. The sorted file can be stored in ascending or descending order.

SEE CHAPTER 8 FOR DETAILS

Appendix 10

GLOSSARY

ACIA (Asynchronous Communications Interface Adapter) An IC used for serial data transfer between a device such as a small computer and a serial terminal.

A/D (Analog/Digital) Refers to changing an analog signal to a digital signal which the computer can use.

BACKPLANE BOARD (Sometimes called Mother Board) Allows simple interconnection between small computer boards using the same bus.

BASIC (Beginners All-Purpose Symbolic Instruction Code) A popular computer language ideally suited for use with Ohio Scientific computers. One of the simplest languages to learn, it can be used for a wide variety of applications.

BAUD A measure of the speed with which information can be communicated between two devices. For example, if the information is in the form of alphabetic characters, then 300 baud usually corresonds to about 30 characters per second.

BIT The smallest amount of information that can be known (one or zero). Eight bits equal one byte.

BUS The means used to transfer information from one part of the computer to another.

BYTE A unit of information composed of 8 bits, which is treated by the computer as a single unit. A byte is usually used to represent an alphanumeric character or a number in the range of 0 to 255.

CASSETTE A compact magnetic tape medium for the electronic storage of data. Most personal computers use ordinary audio-cassette tape recorders and cassette tapes.

CENTRAL PROCESSING UNIT (CPU) The part of computer hardware responsible for interpreting data and executing instructions.

CHIP A small rectangular module which encapsulates an integrated circuit.

COMPUTER An electronic device which is programmable and which processes, operates on, and outputs information according to its stored program upon receipt of signals through an I/O device.

COMPUTER LANGUAGE A language that is used for programming a computer, e.g., BASIC.

CURSOR The marker (underline, rectangle, etc.) on a video monitor screen which indicates the location on the screen where the next character will appear.

DAC (Digital-to-Analog Converter) A device that changes digital signals into one continous analog signal (voltage output).

DATA The information units, or signals, that are processed by a computer.

DIGITAL Word used to described information that can be represented by a collection of bits. Modern computers store information in digital form.

DISK A circular piece of rigid material which resembles a record and which has a magnetic coating similar to that found on ordinary recording tape. Digital information can be stored magnetically on a disk.

DISK DRIVE A peripheral which can store information on, and retrieve information from, a disk. A floppy disk drive can store and retrieve information from a floppy disk.

EPROM (Erasable Programmable Read Only Memory) Information stored in an EPROM IC (Integrated Circuit) can only be removed by special light sources or specific voltages (depending on the type of EPROM). Through the use of a special programming device, the user can store a set of information in the EPROM after it has been erased.

FLOPPY DISK A thin, pliable 8" or 5-1/4" flexible media for storing data. 8" disks store 3, or more, times as much information as 5-1/4" floppies and access the information faster.

FOREGROUND/BACKGROUND Operation term used to describe the ability of a computer to function with normal programs at the same time it monitors external devices, e.g., home appliances, security, etc.

HARD COPY Information printed on paper or any durable surface, as opposed to information temporarily presented on the CRT screen (see Monitor)

HARDWARE The physical equipment that makes up the computer system.

IC (Integrated Circuit) Many miniature electronic components (transistors, diodes, resistors, etc.) built into one small multicontact unit (chip) to produce a special purpose circuit.

I/O (Input/Output) Refers to bringing information into the machine in a form it recognized and allowing the machine to transmit information. In other words, communicating with the

outside world.

INPUT Signals given to a computer for processing.

INTERFACE The connection between two systems. A printer interface, for example, connects the printer to the computer.

JOYSTICK Accessory equipment (peripheral) that permits the user to move the figures on the monitor. For example, when you and another person play a joystick computer game, you operate joysticks to perform the functions of the game.

K The initial "K" stands for "kilo", meaning 1,000. In computer language, K means 1,024 bytes of information that can be stored in a computer system. A computer with 16K memory, for example, has 16 times 1,024, (16,384) bytes of memory.

LSI (Large Scale Integration) Descriptive of the type of circuit in an IC chip where thousands of electronic functions are included.

MEMORY The area in the computer for storage of data and instructions.

MICROCOMPUTER A computer based on a microprocessor.

MICROPROCESSOR The "brains" or CPU of a modern personal computer. All Ohio Scientific personal computers use the 6502 microprocessor, generally recognized as the fastest microprocessor available.

MINI-FLOPPY DISK A small 5-1/4" floppy disk that stores about 1/3 the information on an 8" floppy disk.

MODEM Word derived from MOdulator-DEModulator. A device that allows the computer to communicate over telephone lines and other communications media by changing digital information into audio tones (modulating) and from audio tones into digital information (demodulating).

MONITOR A CRT or television screen. You can purchase an Ohio Scientific monitor to hook up to your computer or else simply use an ordinary TV set and attach it with an RF convertor.

OS Operating system.

PC BOARD (Printed Circuit Board) A card with foils (electronically conductive pathways) connecting electronic components which are mounted on the board.

PERIPHERAL Any device that can send informtion to and/or receive information from a computer, e.g., printer, modem, etc.

PIA Peripheral Interface Adapter. A programmable control IC.

PRINTER A peripheral device which makes hard copy of letters and numbers.

PROGRAM A set of instruction, arranged in a specific sequence, for directing the execution of a specific task, or the solution of a problem, by a computer.

PROM (Programmable Read Only Memory) Memory which can have information stored on it once, but is not normally changeable.

PROMPT A signal given by a computer to indicate that a particular function is ready.

RAM (Random Access Memory) A storage device and main memory of any computer which can be read from and written into. Information and programs are stored in RAM, and they can be retrieved or changed by a program.

ROM (Read Only Memory) A memory storage device in which the information is stored once, usually by the manufacturer, and cannot be changed.

SOFTWARE Programs and operating systems used by the computer; they may be on cassette or on disk and in ROM.

INDEX

C

D

R

S

S - continued...

T

U

Addendum 1

Using Machine Language Subroutines

A. Finding a Place for a Machine Language Routine

There are three areas in memory that are normally used to hold a machine language subroutine:

1) The beginning of workspace

The workspace for BASIC programs begins at address $3A7E (327E in Version 3.2). It is possible, however, to move the starting point for BASIC to a higher address, leaving some free space in front of the BASIC program. One advantage in using this space for a machine language subroutine is that when a BASIC program is in memory and a

```
DISK!"PUT..."
```

is executed, everything in memory is saved from $327E to the end of the BASIC program. Thus, a machine language program in this area is saved along with the BASIC program that uses it. A disadvantage is that the start of the BASIC program will still be at the higher address even after a NEW command, thus effectively shortening the space for a new BASIC program. To restore the workspace to normal the user can either re-boot or LOAD a program known to use the normal workspace. For example, the BEXEC* program on disk #5 of the tutorial set sometimes calls a machine language screen clear routine but

this routine does not change the normal workspace otherwise it would still be there after a NEW command

The user can do some PEEKS to the file headers to determine the addresses of the beginning and end of any BASIC program in the workspace. A five byte file header is maintained at addresses $3A79(=14969) through $3A7D(=14973) ($3279-$327D for 3.2 systems).

Address	Contents
3A79 & 3A7A	A pointer to the beginning of the BASIC program (low, high)
3A7B & 3A7C	A pointer to the end of the BASIC program (low, high)
3A7D	The number of tracks needed to hold the program.

For example, suppose the contents of these locations are:

Address	Contents
3A79	7F
3A7A	3B
3A7B	8Ø
3A7C	4B
3A7D	Ø2

Since the workspace always begin at $3A7F and this program begins at $3B7F, there are $100(=256) bytes free at the beginning of the workspace. The program ends at $4B8Ø and two disk tracks are required to hold the program.

To reserve a place at the beginning of memory for a machine language program, the programmer should

RUN"CHANGE"

He should respond YES to the question

CHANGE BASIC'S WORKSPACE LIMITS?

and also respond YES to

WANT TO LEAVE ANY ROOM BEFORE THE WORKSPACE?

The programmer should create this space after the machine language subroutine has been created (so he knows its size in bytes) but before the BASIC program has been entered. Please refer to CHANGE (pages 141-151) in the Tutorial Manual. If the BASIC program already exists, either in the workspace or on disk, then the indirect files technique must be used. See Chapter 12 of the BASIC Reference Manual. The area at the beginning of the workspace is used as an I/O buffer area for BASIC programs that use disk data files. When a machine language subroutine resides at the beginning of the workspace it is also said to be in a "buffer" although it is not an I/O buffer.

2) The end of workspace

Machine language subroutines can be put at the end of memory. This area of memory is not "safe" until the end of workspace pointers (at 132 and 133) are reset

because even short BASIC programs store generated string data (e.g., through string addition or INPUT statements) at the end of memory. The address of the end of memory is stored at locations 132 (low half) and 133 (high half) and the user can POKE new numbers to these bytes to create a safe place at the end of memory. For example, on a 24K system, the values are Ø at address 132 and 96 at 133 (these are decimal values). Since 96=$6Ø, this means that the end of memory is at address $6ØØØ.

The commands

```
POKE 133,95:POKE 132,128
```

will set this end of memory pointer to $5F8Ø (since 95=$5F and 128=$80) thus creating a 128 byte "safe" area at the end of memory. The numbers at addresses 132 and 133 are not saved with the BASIC program so that the same POKEs must be done every time the program is run.

3) The directory buffer

There is a one page (=256 byte) area between addresses $2E79 and $2F78, inclusive, that is used to hold the disk directory and is used for only that. This directory is brought into memory only when a reference is made to a disk file name, as for example, in the DIR directory program or when DISK!"LOAD. "Filename or RUN"filename" is executed. This area is unused otherwise.

B. Bringing a Machine Language Program into Memory

1) POKEs

A machine language routine may be POKEd into memory by the BASIC program that uses it. This method is used in the BEXEC* program to put a screen clearing routine into the directory buffer. For example, consider the three line assembly language program below which puts an airplane character near the center of the video screen:

```
2E8Ø A9ED   LDA #237
2E82 8D1FD4 STA $D41F
2E85 6Ø     RTS
```

The first column is the hexadecimal address, the next column is the machine language, also in hex, and the last two columns are the source code. Thus, the seven bytes of the machine language program are, in hex and decimal:

```
$A9=169   (LDA)
$ED=237   (ASC Code for airplane character)
$8D=141   (STA)
$1F=31    (screen location, lo-hi
$D4=212     byte format)
$6Ø=96    (RTS)
```

The beginning address is $2E8Ø=11904. A BASIC program to put this machine language into memory is

```
FOR I=Ø TO 5
READ X
POKE 11904+I,X
NEXT I
DATA 169, 237, 141, 31, 212, 96
```

2) DISK!"CALL..."

If a machine language routine has been put into memory by an A3 command to the Assembler/Editor or by hand using the PROM monitor (for details see the Assembler/Editor and Extended Monitor Manual) then it can be saved on disk using

DISK!"SAVE..."

A BASIC program can bring it back to memory with

DISK!"CALL..."

(refer to Tutorial Manual p. 58 for CALL and SAVE details) The user should note that the number of bytes brought into memory by this method is always a multiple of 256.

3) Putting it in the BASIC file

If a machine language routine has been put into a "buffer" area at the beginning of workspace (i.e., between address $3A79 and the start of the BASIC program) then a

DISK!"PUT..."

command will save the buffer area along with the BASIC program. Hence, a machine language subroutine in a buffer will be brought into memory whenever the BASIC program is RUN or LOADed (refer to Tutorial Manual p. 60 for LOAD and PUT details).

C. Calling a Machine Language Subroutine from BASIC

1) Y=USR(X)

This statement branches (via a JSR instruction) to the address stored at memory locations 574 (=$23E) and

575 (=$23F). The low order half of the address is at 574 and the high half at 575. Thus, for example, the three line subroutine given in Section B could be called by

```
POKE 574,128:POKE 575,46:Y=USR(X).
```

These are the required POKEs since the starting address is $2E8Ø and $2E=46 and $8Ø=128. In

```
Y=USR(X)
```

the X may be any formula and Y may be any variable. The calling BASIC program may pass a parameter to the subroutine via the formula X and a built in routine may be called by the machine language subroutine to pass back a valve to the variable Y. See Chapter 13 of the BASIC Reference Manual or Chapter 9 of the Assembler/Editor and Extended Monitor Reference Manual for details. If the subroutine does not call a routine to store a value at Y then Y will have the same value as X after return from the subroutine. When a routine is called by the USR function, no registers must be saved, however, the integrity of pages Ø and 1 must be maintained. Thus, if a machine language program is called via the USR function, any page zero locations that are used may have to be saved and restored.

2) DISK!"GOXXXX"

Here XXXX is the address of the subroutine in hex. This command causes a JSR to be executed. Parameters may be passed using POKEs and PEEKs. (Parameters may also be passed in this manner when the routine is called using the USR function.)

For example, consider the routine below. When called, it will fill the lower 1/8 of the screen with the character whose code is stored at the byte labeled CHR.

```
                          *=$5ØØØ
5ØØØ AD ØC 5Ø             LDA CHR
5ØØ3 A2 ØØ                LDX #Ø
5ØØ5 9D ØØ D6 LOOP        STA $D6ØØ,X
5ØØ8 E8                   INX
5ØØ9 DØ FA                BNE LOOP
5ØØB 6Ø                   RTS
5ØØC                CHR   .BYTE Ø
```

If this routine is in memory then the BASIC statements ($5ØØC=2Ø492, CHR$(36)= $)

```
POKE 2Ø492,36:DISK!"GO 5ØØØ"
```

will cause the subroutine to fill the lower 1/8 of the screen with dollar signs ($).

It can be called by USR as follows:

```
POKE     574,Ø
POKE     575,8Ø
POKE   2Ø492,36
Y=USR(X)
```

Similarly, PEEKs can be used to retrieve answers from

a machine language subroutine.

The execution of a DISK!"GO XXXX" produces the following sequence of events. Page Ø and 1 are swapped (moved) to a temporary storage area at $2E79-$3Ø78. A JSR to address XXXX takes place. The code at XXXX is processed. Upon completion of this code, pages Ø and 1 are swapped back from $2E79-$3078 and the program continues to the next BASIC statement.

Thus, use of the DISK!"GO XXXX" command allows the programmer full use of pages Ø and 1. The trade-off, of course, is that DISK!"GO XXXX" is somewhat slower than the equivalent USR call to a machine code subroutine.