OS-65D V3.0 User's Manual

Table of Contents

Features

*Convenient to use "conventional" disk operating system

*Available for all OSI 6502 mini-floppy and 8" floppy configurations

*Supports 9-Digit BASIC, Assembler/Editor, Extended Machine Code Monitor and transient code programs

*Utilizes named files and manually allocated files interchangeably

*Features convenient to use BASIC oriented sequential and random access data files

*Supports up to four floppy drives

*Supports 430 serial, 550 serial (16 port) parallel printer, cassette and memory I/O as well as serial console and/or keyboard with video console

*Can be directly converted to a locked menu-oriented system for end users

*Contains all OS-65D V2.0 features as a subset and can read version 2 files and assign file names to them

*Supports multiple variable length disk buffers and variable length sectors on diskette

Introduction

OS-65D Version 3.0 is a convenient to use disk operating system which fully supports Microsoft's 9-Digit Extended BASIC, a 6502 resident Assembler/Editor, 6502 Extended Machine Code Monitor and various I/O devices. The operating system is available for all Ohio Scientific mini-floppy and full size floppy disk configurations. The system is convenient for beginners to use via the programming language BASIC. It supports writing programs in BASIC, storing programs on disk, recalling programs and reading and writing sequential and random access data files in BASIC. The system also fully supports assembler language programming for the 6502. In conjunction with its assembler and machine code capabilities, it offers an extensive machine code debugging aid, the Extended Monitor. The system is also well suited to utilize machine code subroutines in conjunction with BASIC programs. It has several advanced features such as variable sector length and the capability of its stand-alone disk operating system kernel to support other languages.

This manual will cover the above features starting with fundamental operation of the system for the BASIC programmer and advancing to more detailed levels. The manual is written to permit the user to fully utilize the computer system at the BASIC language level, without ever having to read those portions of the manual covering assembler level operation. For the user's convenience, a condensed User's Guide that covers all features of OS-65D Version 3.0 is included at the end of this manual.

Using the System in BASIC

Before using any floppy diskettes, please carefully read all the warnings about the care and handling of diskettes and the floppy disk system in the main operator's manual accompanying your computer. Once you have the system properly connected and powered up, place the 65D Version 3.0 diskette, label side up, in the "A" drive of your disk system. There are basically two types of 65D 3.0 diskettes: Development disks and menu-oriented Applications disks. Both boot up directly in the programming language BASIC and execute a BASIC program called BEXEC*. With either type of diskette, the proper procedure is as follows:

1. Place the diskette into the disk drive.
2. Close the drive door.
3. Depress the reset button in front of the CPU or the break key on the computer's keyboard depending on the model of the computer you have.
4. Check to be sure the shift lock key is in the locked or down position on polled keyboard systems.
5. Depress the "D" key. This selects the floppy disk bootstrap which will load the operating system from disk into memory. A series of messages will appear on the screen.

Menu-Oriented Disks

Applications disks display a menu when booted which is a list of numbers and program descriptions, and finally, a message such as "YOUR SELECTION?". To select the desired program, you simply type the number corresponding to the desired selection and depress the RETURN key. The operating system will then load that program and execute it.

Note that all inputs you type into the computer must be followed by pressing the RETURN key. This is referred to as "line-oriented input". It offers a tremendous advantage over character oriented input in that until the RETURN key is pressed, typing errors can be corrected by merely typing a delete character after the error, then typing the correct character. (On various keyboards the delete character (hex code 5F) may be a shift-O, underline or back arrow.) On video terminals with backspace capability the erroneous character is then erased and the cursor is left at the proper position for entry of the correct character. On printing terminals that have backspace capability the erroneous character obviously cannot be erased. However, the print head is left correctly positioned for entry of the correct character. On terminals without backspace capability the cursor/print head is not repositioned but the delete is performed permitting simple correction of errors. As many delete key strokes as needed can be used at any time. For example, if two characters were typed in error, two delete key strokes can be used to eliminate them. In addition to the single character delete, a control-U key entry may be used to delete a whole line. This is done by pressing the U key while holding the CTRL key down.

Menu-oriented operating systems provide operational messages as you go so it is usually not necessary to refer to this manual while operating an Applications disk. It is possible to gain access to the internal software of an Applications disk by typing in the proper response when the menu is displayed. This feature will be covered later, after the user has gained a familiarity with Development disks.

Development Disks

Development disks are specifically for users who wish to write their own programs. Development disks contain utility programs which will provide assistance in developing software instead of providing end user application programs. A Development disk will boot in with a message such as "OS-65D Version 3.0" followed by some other messages and a selection of possible functions, ultimately asking the question "FUNCTION?". The functions in this menu are utility programs which will be covered later.

BASIC and the Immediate Mode

The first objective in mastering 65D Version 3.0 is to learn to utilize the programming language BASIC in the immediate mode and to write simple programs. This is accomplished by selecting a Development disk, booting it in by typing D and answering "UNLOCK" to "FUNCTION?". (Note that the RETURN key must be hit at the completion of each line of input.) This operation initializes BASIC, prepares it for end user programming and returns the user to the BASIC immediate mode displaying the prompter "OK". At this point, the computer will accept almost all standard BASIC statements in the immediate mode. The immediate mode can be used in conjunction with any standard BASIC textbook for mastering the concepts of the programming language BASIC. The following is a short introduction to programming in BASIC and some sample programs that can be run. Once you have mastered elementary programming in BASIC, proceed to the next section which covers loading BASIC programs from disk and storing BASIC programs on disk.

PROGRAM EXAMPLE

The following program example demonstrates some of the more fundamental concepts of BASIC. This program may be entered when the computer replies "OK". Enter the program exactly as it appears, including all punctuation, etc.

```
1Ø  PRINT "HELLO!  I'M YOUR NEW COMPUTER!" <RETURN>
2Ø  PRINT  <RETURN>
3Ø  END    <RETURN>
```

Now, check the program to be sure you have entered it correctly. Type in the word LIST and <RETURN>. This instructs the computer to print out the program as stored within the computer's memory.

```
LIST <RETURN>
```

To have the computer execute ("run") the program, type in:

```
RUN <RETURN>
```

The computer should then print:

```
HELLO!  I'M YOUR NEW COMPUTER!
```

The BASIC language makes it easy to modify (edit) a program. Errors within a line may be corrected by retyping the line. Additional statements may be incorporated into a program by sequencing the new line numbers within the existing program. The following additions to the example program demonstrate these editing concepts.

```
5 FOR X=Ø TO 3Ø  <RETURN>
25 NEXT X  <RETURN>
```

To examine the program as amended, type LIST <RETURN>.
To execute the new program, type RUN <RETURN>.

The computer operating manual contains a more in-depth discussion of BASIC, several sample programs and a reference manual on BASIC.

You may also wish to refer to one of the many BASIC programming texts now available for an in-depth study of BASIC.

Loading, Saving and Running BASIC Programs

OS-65D Version 3.0 allows the user to LOAD, SAVE and RUN BASIC programs specified by starting track number or by up to a six character file name. This unique approach allows maximum versatility in that the user can allow the disk system to locate the space for files or can manually specify exactly where files appear on the disk, as desired.

Floppy Disk Formats

Floppy disks are divided into concentric circles called tracks. Each track can be further divided into entities called "sectors". An 8" floppy disk has 77 usable tracks. Mini-floppy disks have from 35 to 40 usable tracks depending upon the quality of the read/write head in the floppy diskette drive and the quality of the floppy media. Tracks are numbered from 0 up such that the 5th physical track on the disk is track 4. OS-65D Version 3.0 stores BASIC programs starting on track boundaries and uses an integer number of tracks to store each program. That is, it stores programs on a single sector per track. Programs that are multiple tracks in length are stored on contiguous tracks, that is, if a program is 3 tracks long and is specified to be stored on track 40, it is, in fact, stored on track 40, 41 and 42. On 8" floppies, approximately 2800 bytes or characters are stored per track. On mini-floppies, approximately 2000 bytes or characters are stored per track.

Not all of the diskette is available to store user programs. Part of the diskette is occupied by the operating system, the language processors such as BASIC and the Assembler, utility programs

and possibly other end user programs and data files. It is necessary to maintain a directory of what is on the disk both to be able to select desired information from the disk and to know what portions of the disk are available for future storage. For the moment, we will bypass the methods of obtaining directories and proceed to storing a program on diskette and recalling it.

First, type a short program into the computer in BASIC and RUN it. Then, follow the procedure below. Note, when you type EXIT, the system will report the number of tracks required to store the program. On 8" floppies store the program on track 73; on 5" floppies store the program on track 34.

Procedure for Saving a Program on Disk by Track Number

A. After the program has been entered:

1. Type EXIT.(By now you should be remembering to hit the RETURN key after each line of input.)

2. BASIC will report number of tracks needed for storage. Then the DOS prompter A* will appear.

3. Type PUT (track number) where (track number) = 73 for the example on 8" floppies and 34 on 5" floppies.

   WARNING: PUT (track number) will place new programs right over old files on the disk, so be sure that the tracks you specify don't contain other important software (in the example, they don't).

4. Type RETURN BASIC or RE BA in shorthand.

5. The BASIC prompter "OK" should appear with the program still in memory.

Type NEW to clear the program from memory and reinitialize the work space. Now follow the procedure on the next page, specifying track 73 for 8" floppies and track 34 on a 5" floppy.

Procedure for Loading a Program from Disk by Track Number

1. Type EXIT
2. Ignore the track size report BASIC puts out
3. Type LOAD (track number) where (track number) is the starting track of the desired program
4. Type RE BA
5. The BASIC prompter "OK" should appear with the program in memory
6. RUN or LIST the program as desired

The preceding process could be considered tedious for bringing in programs to be run. There is a much shorter way of bringing in programs and running them. This can be demonstrated by typing NEW to initialize the work space and then typing the statement RUN (track number) where (track number) is 73 or 34. This brings the program into the work space and automatically starts executing it.

Utilizing Named Program Files

It is somewhat difficult to have to remember the locations of all programs by track number. For example, it is easy to forget whether a program you want is on track 72 or track 27. Therefore, it is desirable to be able to utilize a name for a program instead of its track number.

To utilize named files on the disk, utility programs which are present on the diskette must be used. These programs are written in BASIC and include DIR, CREATE and DELETE. There are more utility programs, but these are the only ones necessary for saving and recalling named BASIC programs. DIR is the directory program. This program, when executed, lists or prints out a directory of the disk files by name and track utilization. Disk files can include BASIC programs, BASIC data files, assembler source code, machine code and other special files such as the utilities programs. To obtain a disk directory, simple type RUN"DIR while in the BASIC immediate mode. Or type DIR directly to the question"FUNCTION?" when the system is booted. The directory program then asks if you want line printer output instead of console output. It then follows with the directory of file names and track ranges. The following two listings show the standard directory for mini-floppy and 8" floppy Development disks.

Mini-Floppy Disk Directory

OS-65D VERSION 3.0
-- DIRECTORY --

FILE NAME	TRACK RANGE
OS-65D3	Ø-12
BEXEC*	14-14
CHANGE	15-16
CREATE	17-19
DELETE	2Ø-2Ø
DIR	21-21
DIRSRT	22-22
RANLST	23-24
RENAME	25-25
SECDIR	26-26
SEQLST	27-28
TRACE	29-29
ZERO	3Ø-31
ASAMPL	32-32

5Ø ENTRIES FREE OUT OF 64

Full Size Floppy Disk Directory

```
OS-65D VERSION 3.0

 -- DIRECTORY --

FILE NAME    TRACK RANGE
-----------------------
OS65D3 0 - 8
BEXEC* 9 - 9
CHANGE 10 - 10
CREATE 13 - 14
DELETE 15 - 15
DIR    16 - 16
DIRSRT 17 - 17
RANLST 18 - 19
RENAME 20 - 20
SECDIR 21 - 21
SEQLST 22 - 23
TRACE  24 - 24
ZERO   25 - 26
ASAMPL 27 - 27

 50 ENTRIES FREE OUT OF 64
```

The directory listing shows that the program named DIR resides on track 16 so that, in fact, the program could be run on an 8" floppy by the statement RUN"16 just as well as it could be by the statement RUN"DIR. For more information on the directory program and the sorted directory program, DIRSRT, refer to the utilities description portion of the manual.

Saving a BASIC Program on Disk Via a Named File

In order to save a program on disk as a named file, the disk file must exist on the disk and appear in the directory. A file is created on disk by use of the CREATE utility program. This program allows the creation of a disk file of any size from one track to the total free space of the disk. The file must have a six character file name which is unique, that is, the name cannot be the same as that of an existing file. The CREATE utility also checks to make sure that the tracks specified are not in use at the moment to preclude the possibility of over-writing or destroying other data on the disk. To utilize the CREATE program, simply type RUN"CREATE. To start, CREATE a one track long program called TEST. For more detailed information on the CREATE program, refer to the utilities description portion of the manual. Once a file such as the example file TEST has been created with the CREATE utility, you can directly store a program in it. Key in a short program and run it. Then to store this program on disk in the file TEST, type the following statement: DISK!"PUT TEST". This statement saves the program currently in the work space under the file name TEST. If TEST does not exist or you misspell it, the disk operating system will report the error.

Loading a BASIC Program From Disk By File Name

To load and run a BASIC program by file name, use the same procedure as you have used for utility programs. Simple type the statement RUN"TEST". If you want to bring the program into the work space without running it, type DISK!"LOAD TEST". This loads the program into the work space but does not execute it. After these exercises have been completed, you can verify the existence of the file TEST by running the directory program and observing what track it appears on.

Deleting Files

After utilizing a diskette for awhile, it may be desirable to remove a file from the disk because the file is no longer needed or possibly because the program is becoming too large for that particular file and the file must be recreated a larger size. Files can be removed from the directory and subsequently from the disk by use of the Delete Utility. Refer to the utility documentation portion of this manual for instructions on the use of this utility.

Other Useful Features For Loading and Saving Programs on Disk

We have now covered all the fundamentals required to put programs on a diskette and recall them from a diskette. The following discussion will provide additional insights into the use of the disk system for BASIC programs and other files.

Tips for File Use

File names can be up to six characters long and are generally three to six characters. The first character in the file name must be alphabetic and the name cannot include spaces. The

directory program lists out file names as they appear in the directory. For this reason, a sorted directory program, DIRSRT, is available. It sorts the directory in alphabetic order or track number order. The disk also contains a renaming utility called RENAME which allows a file name to be changed.

Tips On File Size

The OS-65D approach to data files requires that the user know how large his file is initially. For programs, this should not be a problem.

To be safe, the user can simply specify a disk file size as large as or slightly larger than the available RAM for BASIC programs. For example, with the mini-disk system with 20K of RAM slightly less than 8K is available for programs, thus, a four track file will handle any program that can be typed into the machine. The user should always maintain a scratch file, usually with the name SRATCH, which is larger than the memory size of the computer or simply have a large block of free tracks. This file or block of tracks can act as temporary storage in several situations. For example, the user types in a program and then remembers that he did not create a file for it. The procedure is to simply store the program in SRATCH, create an appropriate file, reload the program from SRATCH and store it under its proper name. Another case comes up when a BASIC program outgrows its file size. The program is then stored in SRATCH, the old file is deleted and then recreated in a larger size. These procedures will also be valuable for data files which will be discussed later.

Backing Up Files

On computer systems with two or more disk drives, it is recommended that the user periodically recopy his entire disk to a "back up" disk by use of the Copy Utility. The Copy Utility is a machine code utility and is described in the utilities documentation portion of the manual. On single drive systems, the best approach is to back up work by performing all disk file functions on two diskettes. That is, when a new program is being generated, a file for it should be created on two diskettes and then when the program is entered in the machine, it should be saved on both diskettes by storing it on one disk, removing that disk from the system, placing the other diskette in place and storing it in that diskette. This is a somewhat tedious process which is why dual drive systems are popular.

Modifying BEXEC* and Applications Disks

We have now covered enough information to allow the customization of existing Applications diskettes and the creation of new Applications diskettes. All OS-65D Version 3.0 diskettes boot up in BASIC and call in and execute the BASIC program called BEXEC*. On Applications disks, this program contains a menu of available BASIC programs. On Development disks it may contain a menu of some of the utilities. To access the operating system, that is, to unlock an Applications disk such that programs may be listed and modified, the user must type either UNLOCK or PASS to the question "YOUR SELECTION?" depending upon the particular diskette. The system then reports that it is open for modification. By unlocking the Applications diskette and examining the listing of

the menu program, the user can determine where programs are located on the disk. Programs can then be called in via the LOAD command, modified and saved back on disk. Additional programs can be saved on the disk and menu changes can be made as required. The Applications disks do not contain the named file utility programs CREATE, DIR, etc., but can be utilized in conjunction with these programs if they are brought in from a Development disk. Likewise, the user can generate new Applications disks by simply changing BEXEC* on a Development disk as desired for menu and locked operation.

Advanced Features of OS-65D Version 3.0 9-Digit BASIC

The 9-Digit BASIC in OS-65D Version 3.0 contains several extensions to Microsoft 9-Digit BASIC. These extensions provide:

1. Input/output distribution to various devices
2. Interfaces to the disk operating system kernel
3. Extensions for sequential and random access disk data files

We will now discuss each of these extensions in detail.

BASIC I/O Handling

BASIC input and output is performed with the following commands: INPUT, PRINT and LIST. Under OS-65D BASIC, these statements can be utilized in the normal way for input and output to the console device. Also, input/output can be selectively routed from/to various other devices on the system including a terminal, modem or cassette at the serial port, video display, 430 board based UART, memory buffer, line printer, two disk buffers, 16 port serial board and a null device. Input/output can be routed from/to these devices by simply typing a pound sign (#) and the device number (as listed in the table below) immediately following the INPUT, PRINT or LIST command.

Input Devices	Output Devices
1. Serial Port (ACIA)	1. Serial Port (ACIA)
2. Keyboard on 440/540 Board	2. Video on 440/540 Board
3. UART on 430 Board	3. UART on 430 Board
4. Null	4. Line Printer
5. Memory	5. Memory
6. Disk Buffer 1	6. Disk Buffer 1
7. Disk Buffer 2	7. Disk Buffer 2
8. 550 Board Serial Port	8. 550 Board Serial Port
9. Null	9. Null

The following are examples of the use of these statements.

```
INPUT #8,D$
PRINT #4, "LINE PRINTER"
LIST #6
```

For instance, to store a program on cassette that exists on disk, the user simply calls that program into memory and types LIST#1 or LIST#3 depending on which port his cassette interface is connected to. This lists that program on that device. To output to a printer, the user simply types PRINT #4 and the output will be routed to the line printer. Memory output, device 5, is useful for various experimenter situations such as directly displaying information on the 540 video screen without scrolling. This particular application is covered in the Character Graphics Reference Manual. Device 6 and device 7 are memory buffers for use with disk files. The use of these disk file buffers will covered in the following section. Care must be taken not to route input or output to non-existent or turned off peripheral devices since this will cause the computer system to "hang" and will require a reset which may destroy data in memory.

BASIC to DOS Interface

OS-65D Version 3.0 utilizes a stand alone command processor for the disk operating system. That is, disk operation can be performed even if BASIC is not present in memory. Full discussion of the disk operating commands are in another section of the manual and in the User's Guide. We have already covered some of these commands such as LOAD and PUT. The programmer can leave BASIC and enter the DOS command mode by typing EXIT. If he does not

alter the BASIC interpreter in memory or the work space he can return to BASIC by typing RETURN BASIC or in shorthand form RE BA. The user can also execute a single DOS command without leaving BASIC by utilizing the statement DISK!"string" where string is an operating system command. This statement can be part of a BASIC program, thus, allowing the user to conveniently utilize all the disk operating system commands as part of any BASIC program.

Data Files in BASIC

In many applications it is a practical necessity to store many variables in such a way that they can be recalled at a later date. Specifically, after the power has been turned on and off several times. Such a collection of variables is referred to as a data file. There are two fundamental types of data files available under OS-65D Version 3.0; sequential files and random files.

Sequential Data Files

A sequential data file is a file in which information is output to the file sequentially, one item right after another from the beginning to end of the file. To read information from the file one must sequentially input it. Examples of uses for sequential files, would be store a large numeric array or to store information that can be searched sequentially such as names and phone numbers. Let's walk through the process of having a name and phone number in a sequential file. First, a file of adequate length must be created. Then a program must be written which outputs names and phone numbers to this data file. Another program can be written that reads the individual string entries which are, in fact, names and phone numbers and compares them with a target name which is the name a user is searching for. If this name is found in the file, the next string from the file will be the desired phone number. Each file is terminated by an "end of file" marker which the programmer can use or the programmer may utilize other techniques for his own end of file. For instance, in the telephone program, the string "END" could

be utilized as the "end of file" indication. This would be the last string output to the file and could be checked for when inputting information from the file. OS-65D allows the user one or two disk buffers for use with one or two files. This means that the user can have one or two sequential files in use in his program at any given time. These files are referred to as devices 6 and 7. To utilize files as device 6 and 7, obviously one must equate them to physical files on disk. This is done by use of the OPEN command which equates a named file to a particular device number. For example, the statement, DISK OPEN 6,"TEST2" opens the previously created disk file TEST2 and equates it to device 6. Once this statement has been executed, a statement such as PRINT#6,A$ will print the string A$ to the file TEST2. Likewise, information can be input from a file by the statement INPUT#6,B$. When this statement is executed, the next variable in the data file TEST2 will be read into string variable B$. At the end of a program or when one has completed their use of a particular data file, the statement DISK CLOSE,6 should be executed which closes the data file and assures that all updates to the file are made. Two data files may be in use simultaneosly by opening one on device 6 and one on device 7. Then INPUTS and PRINTS to device 6 and 7 can be made interchangeably. More than two data files can be used in a program by simply closing and re-opening files, as needed.

Steps to Using Sequential Data Files

The following steps must be taken to create and fill a sequential file with information.

1. Using the CREATE utility, create a file to hold the sequential output program with a name such as PROG1.
2. Create a data file with a name such as TEST2.
3. Execute the Change Utility by typing RUN"CHANGE. Use the Change Utility to allocate space for one disk buffer at the beginning of the BASIC program. Refer to the section on disk utilities for explicit information on using the Change Utility.
4. When the CHANGE program is complete, the work space has been reconfigured with space allocated for a disk buffer. The program for use of the single disk file should be entered at this time. The following program may be used. It will place four strings in the disk file TEST2.

```
10  DISK OPEN,6,"TEST2"
20  FOR I=1 TO 4
30  PRINT #6,"STRING",I
40  NEXT I
50  DISK CLOSE,6
```

5. Store the program on disk under the name specified in Step 1.
6. Run the program which should output the strings to the disk file TEST2.
7. Use the utility program SEQLST to list out the contents of the data file TEST2. Refer to the utilities portion of the manual for directions.
8. Make the following changes to the program to use it to

list out the file.

```
30  INPUT #6,D$
35  PRINT D$
```

9. Run the modified program. The results should be the same as they were when SEQLST was run.

Random Data Files

In may instances, sequential files become very 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 brought in and the contents of record 72 could be brought in without looking at any of the records in between. OS-65D Version 3.0 supports one random access file at a time as device 6. This can be used in conjunction with an optional sequential file as device 7. The length of individual records within a random access file can be adjusted by the user but are factory set at 128 bytes. There can be any number of individual variable entries within a record of 128 bytes and one record can overflow into the next so that if the user wanted 256 character records for instance, he would just utilize even record numbers. The following example will use the same data file, TEST2, and use it as a random file with a total of ten records. To reuse this

sequential data file as a random file, we must first perform some housekeeping. This housekeeping is performed with the Zero Utility. The Zero Utility erases all information in a file. To accomplish this, type RUN"ZERO. Then specify TEST2 as the file to be erased. A more complete discussion on the Zero Utility function is present in the utilities portion of this manual. After TEST2 has been zeroed, proceed with the following steps.

Steps to Using Random Data Files

1. Create a new program file or utilize the same program file as in the sequential exercise.
2. Execute the Change Utility and allocate space for one disk buffer.
3. Type in the following program:

```
10  DISK OPEN,6,"TEST2"
20  FOR I=Ø TO 9
30  DISK GET,I
40  FOR J=1 TO 2
50  PRINT#6, "STRING";I;J
60  NEXT J
70  DISK PUT
80  NEXT I
90  DISK CLOSE,6
```

4. Save the program under the file name specified in Step 1.
5. Run the program to fill TEST2 with ten records of information.

6. Utilize the random file list utility RANLST to list out the information placed in TEST2. Note that RANLST only lists one string per record so it does not list the second string we wrote to each file record.

7. Modify the original program via the following lines:

```
5Ø  INPUT #6,D$
55  PRINT D$
7Ø  (deleted)
```

8. Execute the modified program to observe the output information. Output information should be the same as was originally placed in the file.

Note that in the above example, an inner FOR loop is used to write each of two strings to each record of the file. Execution of the PRINT statement for each string causes the data followed by a carriage return character to be written to the file. Although the carriage return character occupies a character of file space, its use after each item written to the file greatly simplifies inputting the data. If a record were written as a single long string, commas would have to be written out between each item or the user would have to provide the detailed programming to break the long string into its separate items whenever the string was input. It is much simpler to write each item with a separate PRINT statement. There is also another limitation preventing long strings from being read. The BASIC input buffer is 72 characters long. Consequently, longer strings are truncated on input.

Using the Assembler/Editor

OS-65D Version 3.0 supports an interactive Assembler/Editor. The Assembler/Editor can be brought in by proceeding with the normal boot in procedure to BASIC's immediate mode. Then type EXIT followed by ASM. This brings in the Assembler/Editor and places the computer in the Editor's immediate mode. Assembler/Editor's operation is as specified in the separate Assembler/Editor Manual, except for the extensions to the Assembler covered here. The Assembler/Editor is an extra cost option. The Assembler/Editor utilizes two types of files. Source files which contain the assembler code and optional object files which contain the machine code generated by the assembly. Under OS-65D Version 3.0, source files can be named or specified by track number. Object files can be stored in variable sector format for placement anywhere in memory or can be stored in named file mode if they are set up to reside in the standard work space. In addition, the disk operating system includes an execute object file command (XQT file name) which allows the direct and convenient execution of machine code files providing they are linked to the operating system and reside in the normal work space area. Named files must be created via the BASIC utility before the assembly process is begun. The user has the option of exiting from the Assembler to the DOS for DOS level commands by the use of the EXIT command and returning by typing RE ASM after completing a command. Or, a command can be sent directly to the DOS by simply preceeding it with an exclamation point (!). For example, !LOAD file name

loads a source code file into the assembler's work space and returns control to the Assembler/Editor. Note you can only return to the Assembler if the Assembler is in the transient processor area. Likewise, you can only return to BASIC if BASIC is in the transient processor area. So, if the Assembler was last used, you will have to type the DOS command BASIC to reboot BASIC. If BASIC was last used, you will have to type the DOS command ASM.

Using the Extended Monitor

OS-65D Version 3.0 also includes an Extended Machine Code Monitor for debugging programs at the byte level. This utility is particularly useful for assembler code work. The Extended Monitor can be entered by booting in the system, exiting BASIC by typing EXIT and by typing EM which boots in the Assembler/ Editor and Extended Monitor and leaves the system in the extended monitor command mode. The OS-65D Version 3.0 User's Guide, at the end of this manual, provides a complete list of the Extended Monitor's commands.

System Overview

The OS-65D Version 3.0 is a highly refined super set of the original OS-65D operating system which was first introduced in 1976. Version 3.0 is a compact, highly responsive operating system for BASIC, assembler and machine code programming. It is suitable for all computer system uses except the most demanding business applications where OS-65U and OS-DMS should be utilized.

System Architecture

Version 3.0 utilizes a stand-alone DOS complete with command interpreter. The DOS and command interpreter are part of the DOS kernel and can be utilized without a programming language. In addition to the DOS kernel, the system contains an I/O distributor which supports all standard Ohio Scientific I/O devices and can route input and output through common locations to any combination of these input and output devices. The system supports a transient processor area, specifically for Microsoft BASIC, the 6502 Assembler/ Editor and the Extended Monitor and can be used for any other 6502 language processors which may be installed on the system. The principal source code or object file work space starts at 317E hex for 8" floppies and 327E for mini-floppies. The following memory map shows the overall layout of the system.

System Memory Map

0-FF	6502 Page Zero
100-FF	6502 Stack
200-22FF	Transient Processor Area for BASIC or Assembler or other language processor
2300-3178	OS-65D V3.0 (to 3278 on mini-floppy versions)
2300-265B	I/O Routines
265C-2A4A	Disk Drivers
2A4B-2E78	Operating System Kernel
2F79-3178	Swapper
317E up to BFFF	Source File Work Space (327E up for mini-floppy) Disk buffers when present normally occupy from 317E up, offsetting the work space (327E on mini-floppy versions)

Utility Program

A complete set of utility programs are provided in the OS-65 Version 3.0 for use in creating new files, copying files, printing directories of files or file contents, etc. These programs may be used without any knowledge of their implementation. However, they are all written in BASIC and may be used by the interested reader as sample programs demonstrating various programming and file accessing techniques.

Descriptions of the operation of the utility programs appear on the following pages.

Create File Utility

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?

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.

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.

Change Parameter Utility

This utility program is used to change the system parameters for terminal width and for the work space limits.

The defined terminal width value for the system is used by the BASIC interpreter to provide automatic line rollover when lines longer than the terminal width are output. A carriage return and line feed character are automatically inserted into the output line when it hits the terminal width. Thus, long lines are output as two or more lines rather than a single truncated line. Since some serial terminals and all OSI video systems automatically provide line rollover, you may not need to change this parameter. Note that changing terminal width with this utility program provides only a temporary change. Whenever the system is rebooted or BASIC is cold started (by typing BAS), the terminal width is set back to its default value 132. If you write a BASIC program that requires a different terminal width, then you must run this utility program to appropriately change the terminal width parameter prior to running that BASIC program. Or, you can include into the BASIC program the following commands which setup terminal width (WD is a BASIC variable which must contain the desired terminal width):

```
POKE 23,WD
NC = INT(WD/14)*14
POKE 24,NC
```

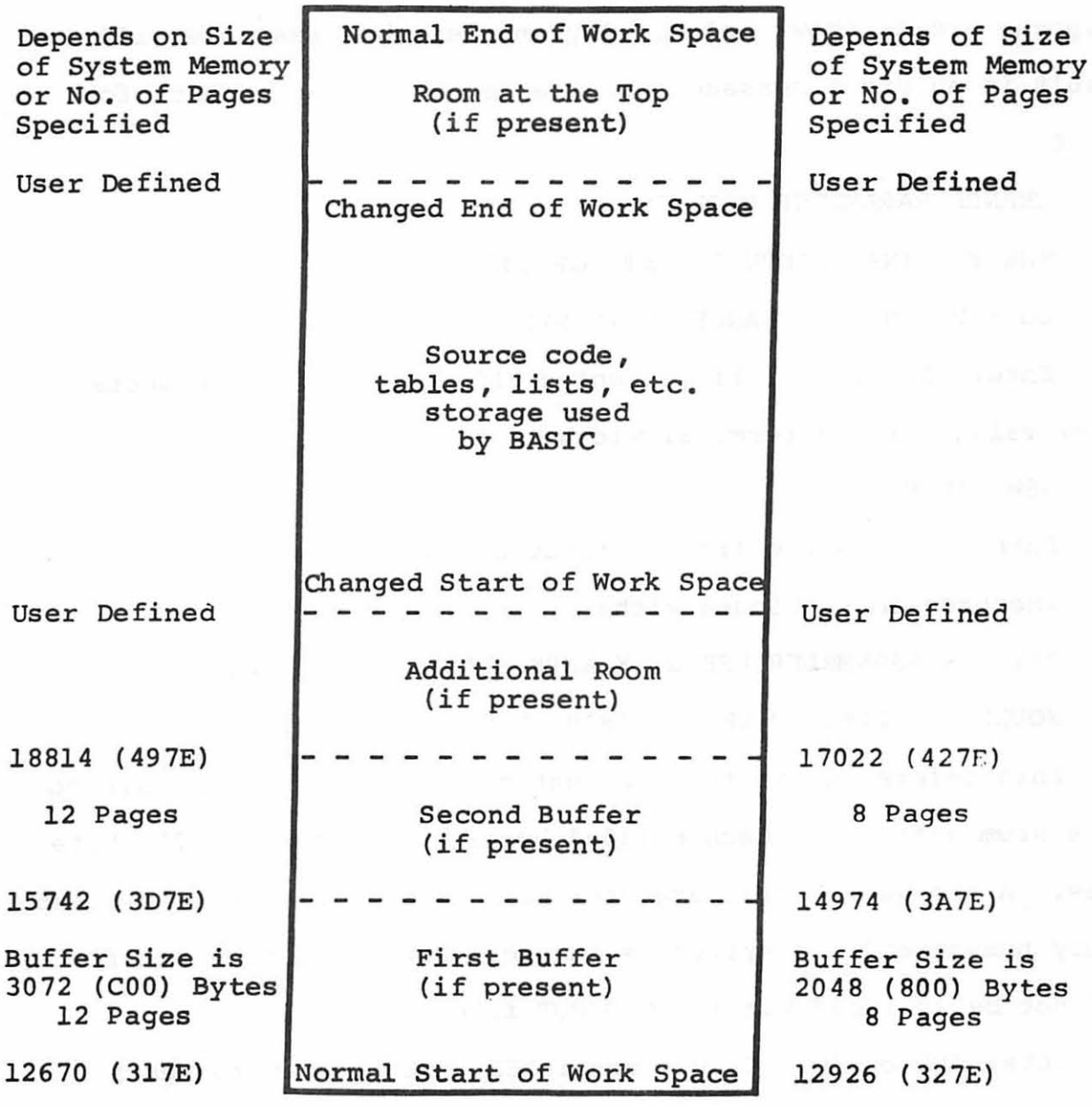
The second POKE, above, sets the column beyond which there are no more 14 character output fields. (Fourteen is the number of character positions allotted to each output field when commas

are used to separate the variables in a PRINT statement.)

The "work space" 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 work space 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 work space is normally the end of the main memory (that memory which starts at address zero and is contiguous up to some higher address).

The BASIC command RUN "file name" and the DOS commands LOAD and PUT provide a means to easily load a disk file into the work space and to put a file that is in the work space back onto disk either by name or by track number. Such files are referred to as LOAD/PUT (or L/P) files.

The Change Parameter Utility Program permits changes to the work space limits so that you can reserve space in a LOAD/PUT file for disk I/O buffers, assembly language object code or whatever. The following diagram shows relevant work space addresses.



OS-65D V3.0 Work Space Addresses in Decimal (Hexadecimal)

To change system parameters, type:

RUN "CHANGE"

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.

CHANGE PARAMETER UTILITY

THE TERMINAL WIDTH IS SET FOR 132

DO YOU WANT TO CHANGE IT (Y/N)?

Enter YES or NO. If you enter YES, the program requests a new value for the terminal width.

NEW VALUE?

Enter a new value from 14 through 255.

The program continues with:

BASIC & ASSEMBLER USE xx K WORK SPACES (yyy PAGES)

WOULD YOU LIKE TO CHANGE THIS (Y/N)?

This refers to the total amount of main memory available to the system software. Each K (1024 bytes) contains four 256 byte pages. A change to this parameter will make a portion of highest memory unavailable to systems software. Note that such memory will not be included within LOAD/PUT files.

Enter YES or NO. If you enter YES, the program requests the number of pages to be used by system software.

HOW MANY PAGES SHOULD THEY USE?

Enter a number of pages from 50 through 191.

The program continues with:

CHANGE BASIC'S WORK SPACE 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 WORK SPACE?

Enter 0, 1 or 2 to reserve that many track buffers at the beginning of the work space. Note that device 6 memory buffered I/O uses the first buffer by default while device 7 uses the second buffer by default. Of course, these defaults can be changed with appropriate POKES. If no buffers are specified, the program asks:

WANT TO LEAVE ANY ROOM BEFORE THE WORK SPACE?

Enter YES or NO. If you enter NO, the program outputs the address of the start of the BASIC work space as shown below. If YES is entered, proceed to the "HOW MANY BYTES?" question below.

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

WANT TO LEAVE ANY ADDITIONAL ROOM?

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 work space.

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

THE BASIC WORK SPACE WILL BE SET TO START AT aaaaa
LEAVING bbbb BYTES FREE IN FRONT OF THE WORK SPACE
IS THAT ALRIGHT?

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

NEW LOWER LIMIT?

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

bbbb BYTES WILL BE FREE BEFORE THE WORK SPACE

The program then continues with:

YOU HAVE xx K OF RAM

DO YOU WANT TO LEAVE ANY ROOM AT THE TOP?

Enter YES or NO. 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 work space and the end of main memory.

The program then outputs:

THE BASIC WORK SPACE WILL BE SET TO END AT ccccc

LEAVING dddd BYTES FREE AFTER THE WORK SPACE

IS THAT ALRIGHT?

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

NEW UPPER LIMIT?

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

eeee BYTES WILL BE FREE AFTER THE WORK SPACE.

Note that the reservation of space <u>after</u> the work space 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 reduced work space that is no longer required.

Program output continues with:

YOU WILL HAVE fffff BYTES FREE IN THE WORK SPACE

IS THAT ALRIGHT?

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

Delete File Utility

This utility program may be used to delete a named file from the directory. This 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 the appropriate password.

The program continues with:

```
FILE NAME?
```

Enter the name of the file to be deleted.

The file will now be deleted from the directory.

Directory Utility

This utility program is used to output a list of all currently existing named files and the numbers of the tracks on which they reside. To output a directory, type:

RUN "DIR"

The program output and the kind of input you may enter in response are as shown below.

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 4; otherwise, it will be on the currently assigned device. If you answer YES and there is no device 4 on the system, the directory will not be output.

A sample directory output appears below.

OS-65D VERSION 3.0

-- DIRECTORY --

FILE NAME	TRACK RANGE
OS-65D3	Ø-8
BEXEC*	9-9
CHANGE	1Ø-1Ø
CREATE	13-14
DELETE	15-15
DIR	16-16
DIRSRT	17-17
RANLST	18-19
RENAME	2Ø-2Ø
SECDIR	21-21
SEQLST	22-23
TRACE	24-24
ZERO	25-26
ASAMPL	27-27

5Ø ENTRIES FREE OUT OF 64

The above directory shows that the system software occupies

tracks zero through eight. OS-65D3 is not a file in the conventional sense, but appears in the directly solely to delineate and reserve the tracks occupied by system software. Track nine contains the BASIC Executive, BEXEC*. This is a BASIC program which always runs when the system is booted and which may be customized as needed to suit your application. In general, tracks ten through 26 contain the various utility programs; however, note that tracks 11 and 12 are free. Track 27 contains the sample assembler language program, ASAMPL.

Sorted Directory Utility

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

RUN "DIRSRT"

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.

SORTED DIRECTORY UTILITY

SORTED BY NAME OR TRACK (N/T)?

Enter N or T 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 4; otherwise, it will be on the currently assigned output device. If you answer YES and there is no device 4 on the system, the directory will not be output.

If neither N or T was entered above

THEN IT WILL BE UNSORTED

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

Sample directory outputs sorted by name and track number appear on the next page.

OS-65D VERSION 3.0

-- DIRECTORY --

FILE NAME	TRACK RANGE
ASAMPL	27-27
BEXEC*	9-9
CHANGE	1Ø-1Ø
CREATE	13-14
DELETE	15-15
DIR	16-16
DIRSRT	17-17
OS-65D3	Ø-8
RANLST	18-19
RENAME	2Ø-2Ø
SECDIR	21-21
SEQLST	22-23
TRACE	24-24
ZERO	25-26

5Ø ENTRIES FREE OUT OF 64

OS-65D VERSION 3.0

-- DIRECTORY --

FILE NAME	TRACK RANGE
OS-65D3	Ø-8
BEXEC*	9-9
CHANGE	1Ø-1Ø
CREATE	13-14
DELETE	15-15
DIR	16-16
DIRSRT	17-17
RANLST	18-19
RENAME	2Ø-2Ø
SECDIR	21-21
SEQLST	22-23
TRACE	24-24
ZERO	25-26
ASAMPL	27-27

5Ø ENTRIES FREE OUT OF 64

Random Access File List Utility

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 question is again asked. To terminate the program, merely type a (return) 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 type a (return) to this question.

Note that this program reads and lists a single string from the start of each record. Random files with more than one entry (an entry is a string of printing characters followed by a return) per record will not be fully listed by this program.

Rename File Utility

This utility program may be used to change the name in the directory of any file listed in the directory. To rename a file, type:

RUN "RENAME"

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.

RENAME UTILITY

OLD NAME?

Enter the name of the file to be renamed as it currently exists in the directory.

The program then outputs:

RENAME "aaaaaa" TO? (aaaaaa is the old name.)

Enter the new name for the file of one to six characters, the first being a letter.

The name will be changed and the utility program will terminate.

Sector Directory Utility

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.

Sequential File Lister Utility

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 100
OK
```

Trace Utility

This utility program may be used to initiate or terminate a BASIC program line number trace. To trace a BASIC program, type:

```
RUN "TRACE"
```

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

```
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 output. 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 include line number outputs while the trace is enabled. This will not adversely affect the operation of the program.

File Zeroing Utility

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 the appropriate password.

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.

OS-65D V3.0 Kernel

The OS-65D V3.0 kernel contains its own command interpreter for handling those commands that are basic to the system. These include commands for initializing diskettes, selecting a disk drive, transferring specific disk sectors and files, initiating various language processors, etc. All kernel commands are listed in the User's Guide with brief descriptions of their function. Those requiring further explanation are also covered below.

Accessing the Kernel

Upon initializing the system, type UNLOCK to the question "FUNCTION?". Then type EXIT. The DOS kernel prompter A* then appears and any kernel commands may be entered.

Changing I/O Distributor Flags

IO nn,mm	Changes input and output flag
IO nn	Changes input flag only
IO ,mm	Changes output flag only

This command changes I/O distributor flags to specify from which device system input is to be taken and to which device or devices system output is to be sent. The values nn and mm shown above in the command are taken from the following table:

nn	Input Device	mm	Output Device
ØØ	Null	ØØ	Null
Ø1	Serial Port (ACIA at FCØØ)	Ø1	Serial Port (ACIA at FCØØ)
Ø2	Keyboard on 440/540 Board	Ø2	Video on 440/540 Board
Ø4	UART on 430 Board	Ø4	UART on 430 Board
Ø8	Null	Ø8	Line Printer
1Ø	Memory	1Ø	Memory
2Ø	Disk Buffer 1	2Ø	Disk Buffer 1
4Ø	Disk Buffer 2	4Ø	Disk Buffer 2
8Ø	550 Board Serial Port	8Ø	550 Board Serial Port

Note that the above values are hexidecimal numbers each of which corresponds to the setting of one bit within the flag byte. Setting no bits in an I/O flag byte specifies the "null device". Output to the null device is thrown away. Input from the null device yields undefined data. If more than one bit is set in the input flag, input is taken from the lowest numbered device (other than null) and the other bits are ignored. More than one bit set in the output flag results in output being sent to each device for which the appropriate bit is set. For example, the command "IO ,Ø9" would result in all output going to both the Serial ACIA Port and the Line Printer.

Some of the above devices need further explanation.

Memory input is from RAM starting at the address contained in locations 238A (low) and 238B (high) with an automatic incrementation of the address after each character is input. Memory output is to RAM starting at the address contained in locations 2391 (low) and 2392 (high) with an automatic incrementation of the address after each character is output. The addresses in these locations can be changed by the user in order to do memory I/O to any available RAM area. The command MEM nnnn,mmmm is provided for this purpose. The nnnn is a four hex digit address

for input, mmmm is an output address.

Disk buffer I/O operates similar to memory I/O described above. However, I/O to the disk buffers also results in automatic disk transfers whenever a buffer (track) boundary is crossed. In order for this disk I/O to properly take place, a few parameters must be set up before performing any of the actual input/output operations. (These parameters are set up in BASIC by the command OPEN.) The parameters and their locations are:

Disk Buffer 1 Locations

2326 (low), 2327 (high)	Buffer start address (normally 317E)*
2328 (low), 2329 (high)	Buffer end address +1 (normally 3D7E)*
232A	First track of file (BCD)
232B	Last track of file (BCD)
232C	Current track in buffer (BCD)
232D	Buffer dirty flag (0 = clean)

Disk Buffer 2 Locations

232E (low), 232F (high)	Buffer start address (normally 3D7E)*
2330 (low), 2331 (high)	Buffer end address +1 (normally 497E)*
2332	First track of file (BCD)
2333	Last track of file (BCD)
2334	Current track in buffer (BCD)
2335	Buffer dirty flag (0 = clean)

Locations of the current buffer addresses are:

Disk Buffer 1 Input	23AC (low) and 23AD (high)
Disk Buffer 2 Input	23FD (low) and 23FE (high)
Disk Buffer 1 Output	23C3 (low) and 23CA (high)
Disk Buffer 2 Output	2416 (low) and 2416 (high)

Proper initialization of these parameters prior to disk I/O includes:

- Setting the current buffer addresses to the buffer end address +1
- Setting the current track in buffer to the first track of file -1

After completing output to disk, the current buffer contents may be left dirty. (Data has been written to the buffer, but the

*Add hex 1ØØ to these addresses in mini-floppy systems.

disk hasn't yet been updated by transferring the buffer out to disk.) If this is so, as indicated by a non-zero buffer dirty flag, the user must perform the final disk transfer. This can be done by reading past the end of the current buffer which will cause a page fault and update the disk.

Transferring Disk Sectors

CALL address=track,sector

SAVE track,sector=address/page

These commands transfer a specified track, sector between RAM and disk. The address must always be four hexidecimal digits, track must be two decimal digits and sector one decimal digit. Pages must be one hexidecimal digit within the range 1-D for full size floppies and 1 through 8 for mini-floppies. A given sector can be referenced only if all lowered numbered sectors exist on the specified track.

NOTE: This version of OS-65D contains more comprehensive disk transfer error checks than previous versions. As a result, under some circumstances, error 9 will be reported when attempting to read or write earlier version diskettes. The D9 command should be entered when this occurs to temporarily defeat the checks for error 9. The system should be reinitialized after completing the transfer to restore error 9 checks.

Executing a Machine Code File

XQT file name

This command loads the file "file name" into the work space at hex 3179 up (3279 up in mini-floppy systems) and transfers control to location 317E (327E). The "file name" can be either the name of

a previously defined file or a track number. Relative location four of the file (which loads into 317D) must contain the number of tracks to be loaded.

Assembly language programs can be developed for use with the XQT command by assembling them with an origin of 317E (327E) and by entering the size of the program in tracks in location 317D (327D) prior to saving the program on disk with the PUT command. Since the Assembler work space also resides at this address, a two-step procedure must be used to create a program with this origin.

1. Assemble the program with an origin of 317E (327E), but with a memory offset (set with the Assembler Mnnnn command) that places the object code into some available memory.

2. Use the Extended Monitor move command to move the program from the selected available memory area to the start of the work space, enter the programs size in tracks then save the program on disk with the PUT command.

For example, with available memory at hex 8000 up, you could use an offset of 5000. The program would then be placed into memory at 817E up (317E + 5000). A complete sample dialog for creating such a program is shown below with user input underlined and explanatory comments.

```
A*ASM                          Loads the Assembler
OSI 6502 ASSEMBLER
COPYRIGHT 1976 BY OSI
.!LOAD file name               Loads the assembler source file
.M5000                         Sets memory offset
.A3                            Assemble object code into memory
.EXIT                          Exit Assembler
A*RE EM                        Enter the Extended Monitor
EM V2.0
:M317E=817E,1111               Moves the object code to work space
:@317D                         Set up size of program in tracks
317D/dd Ø2                     e.g., 2 tracks
:EXIT                          Exit the Extended Monitor
A*PUT file name                PUT machine language program on disk
```

Using Indirect Files

Often it is desirable to be able to merge two or more BASIC or Assembler source files or transfer BASIC programs between incompatible systems such as OS-65D and OS-65U. The Indirect File provides a mechanism for doing this.

In order to use an indirect file, you must have enough RAM to hold the required program(s) in the BASIC or Assembler work space and another copy of the program(s) above the work space. The top of the work space can be appropriately set up with the Assembler Hnnnn command or the BASIC Change Utility Program. Then the indirect file mechanism is set up with this address +1 by entering it into the following locations:

decimal	hex	
9554	2552	Indirect file output address (high)
9368	2498	Indirect file input address (high)

The low part of these addresses is fixed at ØØ.

Transfers to and from the indirect file are then performed as follows:

Dumping Source from the Work Space to an Indirect File

1. Load the source into the BASIC or Assembler work space with the LOAD command.
2. Output the source but type a [after typing LIST or PRINT and before hitting the RETURN key. This turns the indirect file output on.
3. At the completion of the output type a]. This will be echoed as]] and will turn the indirect file output off.

Loading Source from an Indirect File to the Work Space

1. Clear the work space by typing NEW in BASIC or INIZ,Y

in the Assembler. Or, load the source file into the work space into which the indirect file is to be merged.

2. Type a Control-X. The indirect file data will be loaded into the work space. When the] character is loaded at the end of the file, the indirect file input will be automatically terminated.

For normal use, only two operations from the KERNEL mode will be rquired - Initializing Diskettes and Copying Diskettes.

Initializing Diskettes

Once the kernel is entered, a new diskette can be initialized for use by OS-65D V3.0 by removing the operating system disk and placing the diskette to be copied in the "A" drive.

Then type

INT

The machine answers

ARE YOU SURE?

You answer

Y

After the initialization is complete, the prompter A* will re-appear. If an error message is reported during the initialization process, the diskette is probably bad and should be discarded.

```
***************
*  N O T E    *
***************
```

OSI mini-floppy systems have write protect capability. Write protected diskettes have a label covering a notch on one side of the disk. A write protected disk will immediately report an error upon initialization or copying attempts. Simply remove the write protect label before using.

Copying Diskettes

Diskettes can be copied on dual drive systems as follows:

1. First initialize the new diskette as specified above.
2. Place the newly initialized diskette in the "B" (or lower)

drive and the diskette to be copied in the "A" drive.

3. With the KERNEL mode prompter A* on the screen, type

   CA Ø2ØØ=Ø1,2 for 8" floppies or

   CA Ø2ØØ=13,1 for 5" floppies

4. Type

   GO Ø2ØØ

5. The disk copier will appear on the screen. Select 1 and copy from drive "A" to drive "B".

6. Specify from track Ø to 34 on mini-floppies and from track Ø to 76 on 8" floppies.

7. As each track is copied, its track number will appear on the screen.

8. If an error is reported during copying, reinitialize the B diskette and repeat the process. If the error persists, the new diskette is probably bad and should not be used.

NOTE: OS-65D V3.0 can be used to initialize and copy diskettes for all previous versions of OS-65D but not vice versa. In fact, the use of Version 3.0 is recommended over the use of earlier versions for this purpose.

OS-65D Version 3.0 for the I-P

A version of OS-65D V3.0 is available for use with mini-floppies on the OSI I-P Personal Computer. It is identical to that described throughout this manual with the following exceptions:

- the device 4 line printer driver is not included
- the device 3 UART input/output drivers are not included
- only the 440 style video is supported (24 character display) as appropriate to the I-P display
- the device 1 serial ACIA port address is changed to FØØØ as appropriate to the I-P

I-P Pico DOS

A version of OS-65D V3.0 is available as a "Pico-DOS" for use with mini-floppies on the OSI I-P Personal Computer. This system extends the 6-Digit BASIC LOAD and SAVE commands to permit files to be saved on a diskette as well as on the usual cassette.

In order to use the Pico DOS, insert a Pico DOS diskette into the A mini-floppy drive and type a D in response to the D/C/W/M? message. The Pico DOS will boot up with the following message:

```
MINI-65D3 V1.0
MEMORY SIZE? 8955
TERMINAL WIDTH?
```

Note that the memory size has automatically been specified. This is because the Pico DOS occupies memory above this point.

Continue with the initialization by entering terminal width as usual.

The new commands available under the Pico DOS are:

```
LOAD n
SAVE n
```

where n is a program, number 1 through 8.

USER'S GUIDE

OS-65D V3.0 DISK OPERATING SYSTEM

COMMANDS

ASM	LOAD THE ASSEMBLER AND EXTENDED MONITOR. TRANSFER CONTROL TO THE ASSEMBLER.
BASIC	LOAD BASIC AND TRANSFER CONTROL TO IT.
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, V2.0). PLEASE REFER TO COMPATABLITY DISCUSSION LATER.
DIR NN	PRINT SECTOR MAP DIRECTORY OF TRACK "NN".
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. IE. ERASE THE ENTIRE DISKETTE (EXCEPT TRACK 0) 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.
IO NN	CHANGES ONLY THE INPUT FLAG.
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 FILE "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 PROM 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 0.
XQT FILNAM	LOAD THE FILE, "FILNAM" AS IF IT WAS A SOURCE FILE, AND TRANSFER CONTROL TO LOCATION $317E.

NOTE:

- 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 DICTIONARY 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.
CONTROL - U	DELETE ENTIRE LINE AS INPUT.
BACKARROW	DELETE THE LAST CHARACTER TYPED.

ERROR NUMBERS

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!

TRANSIENT UTILITIES

BEXEC* - PROGRAM WHICH GAINS CONTROL ON BOOT IN END USER SYSTEMS.

CHANGE - PERMITS ADJUSTMENT OF THE FOLLOWING:
- TERMINAL WIDTH FOR BASIC.
- THE HIGHEST PAGE OF MEMORY AVAILABLE, WHICH IS WHAT BASIC AND ASM USE WHEN LOADED.
- THE ADJUSTMENT OF THE WORKSPACE LIMITS FOR BASIC. THE RESULT IS A EMPTY WORKSPACE TO THE USER SPECIFICATIONS.

CREATE - ENTER A FILE NAME INTO THE DIRECTORY. AND ZERO OUT THE CREATED FILE ON DISK.

DELETE - REMOVE A FILE NAME FROM DIRECTORY.

DIR - PRINT UNSORTED DISK DIRECTORY.

DIRSRT - PRINT SORTED (BY NAME OR TRACK) DIRECTORY.

RANLST - GENERAL RANDOM ACCESS FILE LIST UTILITY.

RENAME - RENAME A FILE NAME IN DIRECTORY.

SECDIR - PRINT A SECTOR MAP DIRECTORY OF DISK.

SEQLST - GENERAL SEQUENTIAL FILE LIST UTILITY.

TRACE - ENABLE OR DISABLE STATEMENT NUMBER TRACE FEATURE.

ZERO - INITIALIZE CONTENTS OF A DATA FILE TO ZEROS.

I/O FLAG BIT SETTINGS

INPUT:

BIT 0 - ACIA ON CPU BOARD (TERMINAL).
BIT 1 - KEYBOARD ON 440/540 BOARD.
BIT 2 - UART ON 430 BOARD (TERMINAL).
BIT 3 - NULL.
BIT 4 - MEMORY INPUT (AUTO INCREMENTING).
BIT 5 - MEMORY BUFFERED DISK INPUT.
BIT 6 - MEMORY BUFFERED DISK INPUT.
BIT 7 - 550 BOARD ACIA INPUT. AS SELECTED BY "AINDEX" AT LOCATION $2323 (8995 DECIMAL).

OUTPUT:

BIT 0 - ACIA ON CPU BOARD (TERMINAL).
BIT 1 - VIDEO OUTPUT ON 440/540 BOARD.
BIT 2 - UART ON 430 BOAR (TERMINAL).
BIT 3 - LINE PRINTER INTERFACE.
BIT 4 - MEMORY OUTPUT (AUTO INCREMENTING).
BIT 5 - MEMORY BUFFERED DISK OUTPUT.
BIT 6 - MEMORY BUFFERED DISK OUTPUT.
BIT 7 - 550 BOARD ACIA OUTPUT. AS SELECTED BY "AINDEX"

SOURCE FILE FORMAT

RELATIVE DISK ADDRESS	MEMORY ADDRESS	USAGE
0	$3179	SOURCE START (LOW)
1	$317A	SOURCE START (HIGH)
2	$317B	SOURCE END (LOW)
3	$317C	SOURCE END (HI)
4	$317D	NUMBER OF TRACKS REQ.
5 AND ON...	$317E AND ON...	SOURCE TEXT.

DIRECTORY FORMAT

TWO SECTORS (1 AND 2) ON TRACK 8 HOLD THE DIRECTORY. EACH ENTRY REQUIRES 8 BYTES. THUS THERE ARE A TOTAL OF 64 ENTRIES BETWEEN THE TWO SECTORS. THE ENTRIES ARE FORMATTED AS FOLLOWS:

0 - 5 ASCII 6 CHARACTER NAME OF FILE.
6 BCD FIRST TRACK OF FILE.
7 BCD LAST TRACK OF FILE (INCLUDED IN FILE).

MEMORY ALLOCATION

0000 - 22FF	BASIC OR ASSEMBLER/EXTENDED MONITOR.
2200 - 22FE	COLD START INITIALIZATION ON BOOT.
2300 - 265B	INPUT/OUTPUT HANDLERS.
265C - 2A4A	FLOPPY DISK DRIVERS.
2A4B - 2E78	OS-65D V3.0 OPERATING SYSTEM KERNEL.
2E79 - 2F78	DIRECTORY BUFFER.
2F79 - 3178	PAGE 0/1 SWAP BUFFER.
3179 - 317D	SOURCE FILE HEADER.
317E -	SOURCE FILE.

DISKETTE ALLOCATION

0	OS-65D V3.0 (BOOTSTRAP FORMAT LOADS TO 2200 FOR 8 PAGES).
1	SECTOR 1 - REMAINDER OF OS-65D V3.0 (LOADS TO 2A00 FOR 5 PAGES). SECTOR 2 - TRACK ZERO READ/WRITE UTILITY AND COPIER. (LOADS TO 0200 FOR 5 PAGES).
2 - 4	9 DIGIT MICROSOFT 6502 BASIC.
5 - 6	6502 RESIDENT ASSEMBLER/EDITOR.
7	EXTENDED MONITOR.
8	SECTOR 1 - FIRST PAGE OF DIRECTORY. SECTOR 2 - SECOND PAGE OF DIRECTORY. SECTOR 3 - OVERLAY PAGE FOR 9 DIGIT BASIC. SECTOR 4 - PUT/GET OVERLAY FOR 9 DIGIT BASIC.
9 - 76	USER PROGRAMS AND OS-65D UTILITY BASIC PROGRAMS.

9 DIGIT BASIC EXTENTIONS

INPUT PNDSGN<DEVICE NUMBER>,	(INPUT IS SET TO NEW DEVICE, OUTPUT IS SET TO NULL DEVICE IF DEVICE NUMBER > 3, AND NULL INPUTS ARE IGNORED IF DEVICE NUMBER > 3.)

```
INPUT "TEXT";PNDSGN<DEVICE NUMBER>,   <PRINT "TEXT" AT CURRENT
                                       OUTPUT DEVICE, THEN FUNCTION
                                       AS ABOVE>

PRINT PNDSGN<DEVICE NUMBER>,          <PRINT OUTPUT FOR THIS COMMAND
                                       AT NEW DEVICE>

LIST PNDSGN<DEVICE NUMBER>,           <LIST PROGRAM OR SEGMENTS OF
                                       PROGRAM TO NEW DEVICE>

   WHERE <DEVICE NUMBER> FOR OUTPUT IS:

       1 - ACIA TERMINAL
       2 - 440/540 VIDEO TERMINAL
       3 - 430 UART PORT
       4 - LINE PRINTER
       5 - MEMORY OUTPUT
       6 - MEMORY BUFFERED DISK OUTPUT <BIT 5>
       7 - MEMORY BUFFERED DISK OUTPUT <BIT 6>
       8 - 550 ACIA OUTPUT
       9 - NULL OUTPUT

   <DEVICE NUMBER> FOR INPUT IS:

       1 - ACIA TERMINAL
       2 - 440/540 KEYBOARD
       3 - 430 UART PORT
       4 - NULL DEVICE
       5 - MEMORY INPUT
       6 - MEMORY BUFFERED DISK INPUT <BIT 5>
       7 - MEMORY BUFFERED DISK INPUT <BIT 6>
       8 - 550 ACIA INPUT
       9 - NULL INPUT

   AND WHERE PNDSGN IS A POUND SIGN.

EXIT                                  EXIT TO OS-65D V3.0

RUN <STRING>                          LOAD AND RUN FILE WITH NAME IN
                                      <STRING>.

DISK ! <STRING>                       SEND <STRING> TO OS-65D V3.0 AS A
                                      COMMAND LINE.

DISK OPEN,<DEVICE>,<STRING>           OPEN SEQUENTIAL ACCESS DISK FILE
                                      WITH FILE NAME, <STRING>, USING
                                      MEMORY BUFFERED DISK I/O DISTRIBUTOR
                                      DEVICE NUMBER 6 OR 7.  READS
                                      FIRST TRACK OF FILE TO MEMORY AND SETS
                                      UP THE MEMORY POINTERS TO START OF
BUFFER.

DISK CLOSE,<DEVICE>                   FORCES A DISK WRITE OF THE CURRENT
                                      BUFFER CONTENTS TO CURRENT TRACK.

DISK GET,<RECORD NUMBER>              USING LAST FILE OPENED ON THE LUN
                                      6 DEVICE, A CALCULATED TRACK IS READ
                                      INTO MEMORY.  WHERE THAT TRACK IS:
                                      INT(<REC. NUM.>/24)+BASE TRACK GIVEN
                                      IN LAST OPEN COMMAND
```

IT ALSO SETS BOTH MEMORY POINTERS TO:
128*(<REC. NUM.>-INT(<REC. NUM.>/24))
+BASE BUFFER ADDRESS FOR LUN 6 DEVICE.

DISK PUT — WRITE DEVICE 6 BUFFER OUT TO DISK. THE EFFECT IS THE SAME AS A "DISK CLOSE,6".

END USER POKES TO BASIC

LOCATION	OLD	NEW	FUNCTION
2972	58	13	DISABLE , AND : TERMINATORS ON STRING INPUT
2976	44	13	
2073	173	96	IGNORE CONTROL-C
2893	55	28	DISABLE BREAK ON NULL INPUT.
2894	08	11	"REDO FROM START"
741	76	10	REMOVE KEYWORDS, "NEW" AND "LIST"
750	78	10	

OTHER POKES TO BASIC

LOCATION	FUNCTION
23	TERMINAL WIDTH
2888,8722	IF BOTH ARE 0 A NULL INPUT TO A "INPUT" STATEMENT YIELDS AN EMPTY STRING OR A 0. IF BOTH ARE 27 THEN THE INPUT STATEMENT FUNCTIONS AS NORMAL.
8917	USR(X) DISK OPERATION CODE: 0 - WRITE TO DRIVE A 3 - READ FROM DRIVE A 6 - WRITE TO DRIVE B 9 - READ FROM DRIVE B
9826	TRACK NUMBER FOR USR(X) DISK OPERATION
9822	SECTOR NUMBER FOR USR(X) DISK OPERATION
9823	PAGE COUNT FOR USR(X) DISK WRITE, OR NUMBER OF PAGES READ IN BY DISK READ
9824	LOW BYTE OF ADDRESS OF MEMORY BLOCK FOR USR(X) DISK OPERATION
9825	HIGH BYTE OF ADDRESS OF MEMORY BLOCK FOR USR(X) DISK OPERATION

8954 LOCATION OF JSR TO A USR FUNCTION. PRESET TO JSR $22D4. IE. SET UP FOR USR(X) DISK OPERATION

8993 I/O DISTRIBUTOR INPUT FLAG

8994 I/O DISTRIBUTOR OUTPUT FLAG

8995 INDEX TO CURRENT ACIA ON 550 BOARD. IF NUMBERED FROM 0 TO 15 THE VALUE POKED HERE IS 2 TIMES THE ACIA NUMBER.

8996 LOCATION OF A RANDOM NUMBER SEED. THIS LOCATION IS CONSTANTLY INCREMENTED DURING KEYBOARD POLLING

8960 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

9098 LOW BYTE ADDRESS FOR MEMORY INPUT
9099 HIGH BYTE ADDRESS FOR MEMORY INPUT

9105 LOW BYTE ADDRESS FOR MEMORY OUTPUT
9106 HIGH BYTE ADDRESS FOR MEMORY OUTPUT

9132 LOW BYTE ADDRESS FOR MEMORY BUFFERED DISK INPUT
9133 HIGH BYTE ADDRESS FOR MEMORY BUFFERED DISK INPUT
BIT 5 DEVICE. DEFAULTS TO $317E.

9155 LOW BYTE ADDRESS FOR MEMORY BUFFERED DISK OUTPUT
9156 HIGH BYTE ADDRESS FOR MEMORY BUFFERED DISK OUTPUT
BIT 5 DEVICE. DEFAULTS TO $317E.

9213 LOW BYTE ADDRESS FOR MEMORY BUFFERED DISK INPUT
9214 HIGH BYTE ADDRESS FOR MEMORY BUFFERED DISK INPUT
BIT 6 DEVICE. DEFAULTS TO $3D7E.

9238 LOW BYTE ADDRESS FOR MEMORY BUFFERED DISK OUTPUT
9239 HIGH BYTE ADDRESS FOR MEMORY BUFFERED DISK OUTPUT
BIT 6 DEVICE. DEFAULTS TO $3D7E.

8998 MEMORY BUFFERED DISK I/O BIT 5 DEVICE PARAMETERS:

8998-8999	- BUFFER START ADDRESS ($317E)
9000-9001	- BUFFER END ADDRESS ($3D7E)
9002	- FIRST TRACK OF FILE
9003	- LAST TRACK OF FILE
9004	- CURRENT TRACK IN BUFFER
9005	- DIRTY BUFFER FLAG (0=CLEAN)

9006 MEMORY BUFFERED DISK I/O BIT 6 DEVICE PARAMETERS:

9006-9007	- BUFFER START ADDRESS ($3D7E)
9008-9009	- BUFFER END ADDRESS ($497E)
9010	- FIRST TRACK OF FILE
9011	- LAST TRACK OF FILE
9012	- CURRENT TRACK IN BUFFER
9013	- DIRTY BUFFER FLAG (0=CLEAN)

12042 LOCATION OF THE 24 USED BY THE RANDOM ACCESS FILE CALCULATION ROUTINES. THIS LOCATION SHOULD ONLY BE ALTERED AFTER THE OPEN HAS OCCURRED FOR THE RANDOM ACCESS FILE BECAUSE THE PUT GET CODE IS LOAD-

ED INTO THE DIRECTORY BUFFER. THIS IS WHERE THIS
24 RESIDES. MAKING IT A 48 GIVES ONE 64 BYTE RECORDS.

9368 HIGH BYTE ADDRESS FOR INDIRECT FILE INPUT (LOW=00)

9554 HIGH BYTE ADDRESS FOR INDIRECT FILE OUTPUT (LOW=00)

EXTENTIONS TO ASSEMBLER

E EXIT TO OS-65D V3.0.

H<HEX NUM> SET HIGH MEMORY LIMIT TO <HEX NUM>.

M<HEX NUM> SET MEMORY OFFSET FOR A3 ASSEMBLY TO <HEX NUM>.

!<CMD LINE> SEND <CMD LINE> TO OS-65D V3.0 AS A COMMAND TO BE EXECUTED AND THEN RETURN TO ASSEMBLER.

CONROL-I TAB 8 SPACES. ALSO:

- CONTROL-U 7 SPACES.
- CONTROL-Y 6 SPACES.
- CONTROL-T 5 SPACES.
- CONTROL-R 4 SPACES.
- CONTROL-E 3 SPACES.

CONROL-C ABORT CURRENT OPERATION

EXTENDED MONITOR

!TEXT SENT "TEXT" TO OS-65D V3.0 AS A COMMAND.

@NNNN OPEN MEMORY LOCATION "NNNN" FOR EXAMINATION.
SUBCOMMANDS:
- LF - OPEN NEXT LOCATION.
- CR - CLOSE LOCATION.
- DD - PLACE "DD" INTO LOCATION.
- " - PRINT ASCII VALUE OF LOCATION.
- / - REOPEN LOCATION.
- UPARROW - OPEN PREVIOUS LOCATION.

A PRINT AC FROM BREAKPOINT.

BN,LLLL PLACE BREAKPOINT "N" (1-8) AT LOCATION, "LLLL".

C CONTINUE FROM LAST BREAKPOINT.

DNNNN,MMMM DUMP MEMORY FROM "NNNN" TO "MMMM".

EN ELIMINATE BREAKPOINT "N".

EXIT EXIT TO OS-65D V3.0.

FNNNN,MMMM=DD FILL MEMORY FROM "NNNN" TO "MMMM"-1 WITH "DD".

GNNNN	TRANSFER CONTROL TO LOCATION "NNNN".
HNNNN,MMMM<OP>	HEXDECIMAL CALCULATOR PRINTS RESULT OF "NNNN"<OP>"MMMM" WHERE <OP> IS + - * /
I	PRINT BREAK INFORMATION FOR LAST BREAKPOINT
K	PRINT STACK POINTER FROM BREAKPOINT
L	LOAD MEMORY FROM CASSETTE.
MNNNN=MMMM,LLLL	MOVE MEMORY BLOCK "MMMM" TO "LLLL"-1 TO LOCATION "NNNN" AND UP IN MEMORY
NHEX>NNNN,MMMM	SEARCH FOR STRING OF BYTES "HEX" (1-4) BETWEEN MEMORY LOCATION "NNNN" AND "MMMM"-1.
O	PRINT, OVERFLOW/REMAINDER FROM HEX CALCULATOR.
P	PRINT PROCESSOR STATUS WORD FROM BREAKPOINT.
QNNNN	DISASSEMBLE 23 LINES FROM LOCATION "NNNN" A LINEFEED CONTINUES DISASSEMBLY FOR 23 MORE
RMMMM=NNNN,LLLL	RELOCATE "NNNN" TO "LLLL"-1 TO LOCATION "MMMM".
SMMMM,NNNN	SAVE MEMORY BLOCK, "MMMM" TO "NNNN"-1 ON CASSETTE.
T	PRINT BREAKPOINT TABLE.
V	VIEW CONTENTS OF CASSETTE.
WTEXT>MMMM,NNNN	SEARCH FOR ASCII STRING "TEXT" BETWEEN "MMMM" AND AND "NNNN"-1.
X	PRINT X INDEX REGISTER FROM LAST BREAK
Y	PRINT Y INDEX REGISTER FROM LAST BREAK.

NOTE: ALL COMMANDS ARE LINE BUFFERED BY OS-65D.
THUS ONLY 18 CHARACTERS PER LINE ARE ALLOWED
AND CONTROL-U AND BACKARROW APPLY.

DISKETTE COPIER

THE DISKETTE COPY UTILITY IS FOUND ON TRACK 1 SECTOR 2. IT SHOULD BE LOADED INTO LOCATION 200 WITH A "CA 0200=01,2". TO START IT TYPE, "GO 0200". TO SELECT THE COPIER TYPE A "1". THE COPIER AUTOMATICALLY FORMATS THE DESTINATION DISKETTE BEFORE WRITING ON IT

TRACK 0 READ/WRITE UTILITY

THIS UTILITY PERMITS THE READING OF DATA ON TRACK 0 ANYWHERE INTO MEMORY. ALSO THE CAPABLITY IS AVAILABLE TO WRITE ANY BLOCK OF MEMORY TO TRACK 0 SPECIFYING A LOAD ADDRESS AND PAGE COUNT.

THE TRACK ZERO FORMAT IS AS FOLLOWS:

- 1 MILLISECOND DELAY AFTER THE INDEX HOLE.
- THE LOAD ADDRESS OF THE TRACK IN HIGH-LOW FORM.
- THE PAGE COUNT OF HOW MUCH DATA IS ON TRACK ZERO.

TRACK FORMATTING

THE REMAINING TRACKS ARE FORMATTED AS FOLLOWS:

- 1 MILLISECOND DELAY AFTER THE INDEX HOLE.
- A 2 BYTE TRACK START CODE, $43 $57.
- BCD TRACK NUMBER.
- A TRACK TYPE CODE, ALWAYS A $58.

THERE CAN BE ANY MIXTURE OF VARIOUS LENGTH SECTORS HEREAFTER. THE TOTAL PAGE COUNT CAN NOT EXCEED 12 PAGES IF MORE THAN ONE SECTOR IS ON ANY GIVEN TRACK. 13 PAGES CAN BE PLACED ON A TRACK IF ONLY ONE SECTOR RESIDES ON A TRACK. EACH SECTOR IS WRITTEN IN THE FOLLOWING FORMAT:

- PREVIOUS SECTOR LENGTH (4 IF NONE BEFORE) TIMES 800 MICROSECONDS OF DELAY.
- SECTOR START CODE, $76.
- SECTOR NUMBER IN BINARY.
- SECTOR LENGTH IN BINARY.
- SECTOR DATA.

COMPATABILITY WITH EARLIER OS-65DS

THE EARLIER VERSIONS OF OS-65D (IE. EARLIER THAN 3.0) HAD A QUIRK OF OPERATION. WHEN THEY ATTEMPTED TO DO A READ THE HEAD WAS LOADED AND THE ACIA INITIALIZED AT THE RISING EDGE OF THE INDEX HOLE. SINCE THE EARLIER 65D'S FORMAT INCLUDED NO GAP AFTER THE INDEX HOLE, THE ACIA MAY BE INITIALIZED IN THE MIDDLE OF A BYTE. THIS WOULD SET THE ACIA OUT OF SYNC WITH THE DATA. IT WOULD THEN TAKE SEVERAL REVOLUTIONS OF THE DISKETTE BEFORE THE ACIA GOT BACK IN SYNC AND THE TRACK HEADER FOUND. FOR THIS REASON THERE MAY BE PROBLEMS IN READING EARLIER VERSION FILES. THE ERROR ENCOUNTERED IS ERROR 9. THIS ERROR INDICATES THAT THE TRACK HEADER WAS NOT FOUND IN ONE REVOLUTION. SO THAT EARLIER VERSION FILES CAN BE COPIED OVER TO THE NEW SYSTEM, THE D9 COMMAND IS AVAILABLE. IT PREVENTS THE ERROR 9 ERROR CHECKING.