P71

OS-65U™ V1.1

OPERATORS MANUAL

DIREC* IS D 25088
SHOULD BE D 3584

POKE 9899, 98
9898, 14

June 1978

NOTICE

Changes from OS-65U Version 1.0 to Version 1.1 that may Affect Compatibility

The following utility programs have changed and should be used in preference to previous versions. Program and data file formats are unchanged.

BEXEC*

CREATE

PACKER

COPIER

COPYFI

The line printer, device #5, now has software page control. This feature may be disabled with a POKE 14457,66.

Flags 20-23 for disk checksum checking and input escape were changed to 19-22 for compatibility with other flags.

The access rights on all utility programs was changed to WRITE, i.e., they are RUN ONLY without giving the proper password.

A special version of OS-65U is available with a P after the version number. In this version, the keyboard input, device #2, supports the polled keyboard not the older model standard keyboard. Also, the video output, device #2, supports the 540 video board only and not the 440 and 540.

NOTE: Please carefully study the entire manual before proceeding to actually using the accompanying diskette.

Table of Contents

Appendices

OS-65U V1.1, Level I Release Notice

Version 1.1 of OS-65U is the first official release. It contains a number of enhancements that provide additional new features as well as improvements in existing functions. New features include:

1). Multi-terminal operation with up to 16 additional terminals and a timesharing executive program, MULTI, that provides program file load, save and print functions to up to 16 intelligent terminals (2P's and 8P's) connected to a host computer running OS-65U. The Multi-Terminal Operations Manual is available separately.

2). New utility programs:

RENAME for changing filenames and passwords.

FPRINT prints or displays entries in a sequential or random file.

COPYFI for high speed copying from one file to another.

FDUMP prints or displays blocks of data from a file.

CHANGE for changing bytes on disk memory.

LOAD32 and LOAD48 for installing machine language routines as part of BASIC programs.

3). A BASIC statement trace that displays each source line of a BASIC program as it is executed.

4). The ability for BASIC programs to, optionally, retain control for programming end of file action.

5). The ability for BASIC programs to, optionally, transfer control to line 50000 for handling of any disk I/O errors.

6). Line printer page control for automatic paging of line printer output.

7). The ability to limit 440/540 and serial terminal output to a page (screen) at a time.

And the following improvements in existing features are provided:

1). The disk directory is always reloaded by OPEN and LOAD commands to permit diskettes to be interchanged in a drive without rebooting or running the DIR program.

2). Floppy disk drivers now permit increased tolerances on disk drives, thus increasing overall system reliability.

3). The RUN command now permits specification of a starting line number when a filename is specified, e.g., RUN"ACCNTS",50.

4). More comprehensive file access rights, including run-only, are provided.

5). The NEW command clears the name of the last LOADed file, thus preventing use of SAVE without a filename after NEW has been typed.

6). The keyboard driver has been upgraded to support the polled keyboard used with 540 video boards. This is available in a special version of OS-65U with a P after the version number. The Polled Keyboard PROM is required at $FDØØ to support the polled keyboard.

License terms for OS-65U

Ohio Scientific's OS-65U is copyrighted by Ohio Scientific, Inc. and Microsoft, Inc. Any duplication or redistribution of other than original factory copies of OS-65U other than for use on an individual computer system is strictly prohibited. Ohio Scientific dealers and representatives are authorized only to restore destroyed copies of OS-65U on company-provided and serialized diskettes, which contain the Ohio Scientific label and the Microsoft copyright, and clearly specify OS-65U Version 1.0 or higher. Copying and redistribution of OS-65U on any medium other than factory-generated OS-65U diskettes is strictly prohibited and in violation of copyright laws. Each diskette containing OS-65U bears a unique serial number which will be maintained on file at Ohio Scientific, both to protect the software, and to serve as a means of notifying software package owners of updates in the form of corrections of any bugs or improvements that may occur.

Purchasing one copy of OS-65U entitles the user to utilize OS-65U on one system only. The serial number of the computer as well as the license number of the software are to be registered at Ohio Scientific, Inc. The license number is needed to obtain assistance and updates to OS-65U as well as to purchase applications software from dealers based on OS-65U.

WARRANTY

There is no warranty, either expressed or implied, for OS-65U. It has been extensively tested at Ohio Scientific with a battery of standard library programs, and is believed to be reasonably error-free. We do not guarantee the programs in any fashion, nor do we guarantee that all bugs or problems that may occur will be fixed. We will, however, under normal circumstances make an attempt to help you with any problems that may occur, and intend to support this software package whenever possible. If you have any problems with the software or think you have found a bug, please contact us. In the case of any complicated bug, please include a documented example of the problem.

OS-65U License Form

OS-65U License Number 2337

Factory Issue Date 9-19-78

Diskette Serial Numbers 2337

2338

System Serial Number

Keep This For Your Records

OS-65U is a nine digit BASIC language system which includes a complete named file disk operating system for floppy disks and for large capacity, hard surface disks. The file system supports both data and program files.

Data files appear to the user as a single contiguous block of data bytes that can be read or written sequentially or randomly. High speed file searches are possible with a special command that locates a specified character string.

Data file commands include OPEN, PRINT, INPUT, FIND, INDEX and CLOSE.

Program files hold BASIC programs in tokenized, ready-to-run form for higher execution speed.

Program file commands are LOAD, SAVE, RUN and LIST.

Peripheral interface is simplified in OS-65U by a comprehensive set of I/O drivers for:

- Serial ASCII terminals (printer or CRT)
- Low and high density alphanumeric and graphic video displays
- Stand alone keyboards
- Line printers
- Cassette tape recorders
- Up to 16 additional serial terminals

A common I/O distributor permits easy selection of any I/O device or combination of output devices.

OS-65U includes a timesharing executive program that provides file load, save and print functions to up to 16 intelligent terminals (C2-4P's and C2-8P's). This distributed processing

allows each of the 16 users to access floppy disk or hard disk storage for their BASIC programs and to share a central printer for hard copy listings. A Multi-Terminal Operations Manual is available from OSI.

OS-65U includes a comprehensive set of utility programs for management and maintenance of program and data files:

CREATE is used to create new files.

DIR displays or prints a directory of files.

COPIER initializes new disk surfaces and copies from one disk to another.

DELETE deletes a file.

RENAME changes the name of a file.

CHANGE permits machine language changes to disk memory.

COPYFI copies the contents of one file to another.

LOAD32 and LOAD 48 facilitate entry of machine code routines as part of BASIC program files.

FPRINT prints the contents of sequential or random files.

FDUMP dumps complete file contents.

OS-65U also includes a number of extensions to BASIC that provide complete systems programming capability to those who will be adding end-user oriented applications software. These extensions include programmer control of disk error action, comprehensive file access rights control, a "money mode" numerical output format, and a BASIC statement trace to speed debugging.

OS-65U may be stored on either floppy disk or C-D74 disk and may be bootstrap loaded from either. Boot loading from floppy disk may be done with the 65F bootstrap ROM - the same ROM as has been used with OS-65D. In order to boot load from the C-D74 disk, however, both the new 65F3 ROM and the 65H boot ROM are required. (The 65H ROM is at FDXX)

When the reset pushbutton on the computer front panel is pressed the 65F3 ROM outputs the message:

H/D/M?

To boot load from the hard disk (C-D74) type an H.

To boot load from the floppy disk,type a D.

To enter the machine language monitor, type an M.

If either H or D are typed, the OS-65U system will be loaded from disk and initialized. This initialization procedure determines the amount of memory available for BASIC programs and initializes certain peripheral devices. If a peripheral device is present and has power applied but does not respond to the initialization within a reasonable amount of time (up to thirty seconds), a notification message will be output by the initialization program. Examples of these messages are:

DEV B NOT READY

DISK E FAILED

At the completion of system initialization, the title and revision level of the operating system are output and the BASIC interpreter is initiated:

OS-65U V (Version number)

FUNCTION

The user can type

UNLOCK

DIR

PDIR

or any other string

at this time.

UNLOCK enables exiting a program with a carriage return in response to an input statement and enables CNTRL C. The work space is cleared and the prompter "OK" is output. All FLAGs are in the inactive state (odd numbered FLAGs).

DIR displays the disk directory.

PDIR prints the disk directory on the line printer.

Any other string causes the work space to be cleared and the prompter "OK" to be output. Programs can be entered and/or RUN in this mode but can not be exited from hence the system is "LOCKED". All FLAGs are in the inactive state (odd numbered FLAGs) except FLAG 21.

The INITIALIZATION function is performed by BEXEC* which is a normal BASIC program which can be customized. The system can be re-"LOCKED" at any time by running BEXEC*.

C-D74 Master and User System Initialization

Initialization from a C-D74 disk continues with the output of the master system directory.* This is a directory of the available user systems on the disk and might appear as follows:

MASTER SYSTEM DIRECTORY

NO.	NAME	STARTING ADDRESS	LENGTH	ENDING ADDRESS
1	MASTER	0	215040	215040
2	USER1	215040	430080	645120
3	USER2	645120	2150400	2795520

SYSTEM NO.?

Enter the number of the system to be selected.

PASSWORD ?

Enter the password.

The selected system will be initialized, will display its title and enter the BASIC interpreter:

```
OS-65U V1.0
USER1 SYSTEM
OK
```

This only applies when multiple systems have been installed on the C-D74. This function is implemented by linking BEXEC to SYSDIR with appropriate customization of SYSDIR.

How To Customize Your System

The ability to customize BEXEC*, the system initialization program, provides the user programmer the means to easily design a system that suits his particular application requirements.

A review of the supplied version of BEXEC* is provided here to show the types of functions that may be done in this program. Refer to the listing of BEXEC* in the back of this manual while reading the discussion below.

When a 65U system is booted, the monitor PROM is tested to determine which type of console device is used on the system. Location 11664 and 11665 are then set to the device numbers of the console input and output devices. These locations are referenced, thereafter, for the console device number. BEXEC* reads these locations and sets the I/O distributor device flag bytes to set the console as the active I/O device.

BEXEC* also executes FLAG 21 to prevent the operator from terminating the running program by typing only a carriage return in response to an INPUT statement. This forces the operator to choose one of the alternatives intended for him. Also, to prevent the operator from terminating the running program, the Control-C and Control-O functions are disabled with two POKEs.

BEXEC* then prints the name and version number of the operating system. Obviously, any other title or instructions can be output here.

When OS-65U is booted from a floppy disk but also has access to a hard disk, the hard disk cylinder address of the

active hard disk system must be stored in locations 61438 and 61439. The statement at line 150 does this.

BEXEC* then permits the operator to display or print a directory or unlock the system. If a directory is chosen or some other response is given by the operator, the system is left locked. If "UNLOCK" is input then FLAG 22 is executed to permit input escape and locations 14639 and 2073 are POKEd to permit use of Control-C and Control-O.

Many other features can be added to BEXEC* to customize it for a particular application. For example, the system is preset for a terminal width of 132 characters. This can be changed with the following two POKES:

```
POKE 23,TW : POKE 24, INT (TW/14)*14+1
```

When the system is initialized, memory size is established by searching for the end of memory or by using a constant value entered onto disk. Memory size can also be changed in BEXEC* by executing the following:

```
POKE 132,MS-INT(MS/256)*256 : POKE 133,INT(MS/256) : CLEAR
```

Refer to the sections of this manual entitled FLAG commands and Peripheral Device Numbers for other options that can be invoked by BEXEC*.

OS-65U File System Commands

The commands available in Ohio Scientific's Nine Digit BASIC for random and sequential disc file manipulation are described on the following pages. In these descriptions, commands are shown in capital letters and command variables in lower case. The following command variables are string literals (in quotation marks) or string variables:

Variable	Contents
Unit	A,B,C,D,E,F,G, or H
File Name	Up to 6 characters
Password	Up to 4 characters

One additional variable, channel, is a numeric constant or variable of value 1 through 8.

When string literals are used in commands, the final quotation mark may be omitted for brevity. For example, the command DEV"E" and DEV"E are equivalent.

Passwords are not needed in OPEN, RUN or LOAD commands if no password was used at program's create time. Furthermore, passwords are not necessary if the files access rights will not be violated, i.e., if a BASIC program is of the R/W type, passwords are never needed. In general, Ohio Scientific demonstration programs DO NOT USE passwords.

System Commands

DEV unit

Establishes the specified device as the currently selected mass storage unit. All file system commands executed after this command will access the unit last specified in a DEV command for directories as well as for all other files.

The unit specifications are A, B, C or D for floppy discs and E, F, G or H for hard discs.

Data File Manipulation Commands

OPEN file name, password, channel (password optional)

Opens the specified file for access and assigns it to the specified channel number 1-8. The file will then be available for read, write or read and write access as defined when the file was created. If the correct password is given, the file will be available for read and write access regardless of the access rights defined when the file was created.

When a file is opened, the INDEX variable for the assigned channel is set to zero.

CLOSE channel

Closes the file associated with the specified channel or all open files if no channel number is specified in the CLOSE statement. Closing a specific closed channel is an error.

PRINT % channel,...

Outputs data to the file associated with the specified channel. The specified channel must have been previously assigned to a file with the OPEN command and write access permission must exist.

INPUT # device, "message";%channel,...

Inputs data from the file associated with the specified channel. If the optional message is included, it and any further prompting messages are output to the console device, or to the device specified by the optional device specification, if any (see Input/Output Device Specification, page 22. Further prompting messages include a question mark, REDO FROM START and EXTRA IGNORED. If the optional message is omitted,

it and all further prompting messages are suppressed. Input is from the file associated with the channel number which immediately precedes the input variable list. The specified channel number(s) must have previously been assigned to a file with the OPEN command and appropriate read or write access permission must exist.

INDEX ⟨channel⟩ = formula

Sets the value of the INDEX variable for the specified channel to the result of the formula. The index is a positive, byte (character) index into the file assigned to the specified channel. The first character in a file is at index location zero, the second at index location one, etc. When an INPUT or PRINT statement is executed, the data is input or output starting with the character at the index location and proceeding, character by character, to higher index numbers. At the completion of an INPUT or PRINT statement, the index variable points to the next character to be read or written; but, of course, may be altered by the use of the INDEX ⟨channel⟩ command.

There are no limitations on changing an index except that it cannot exceed the length of the associated file. Thus, both random and sequential access methods may be used in file I/O.

The value of an index may also be read by use of the INDEX (channel) function. For example, the statement

INDEX ⟨1⟩ = INDEX (1) + 10

increments the index by ten.

FIND "string", channel

Searches for the specified string in the file assigned to the specified channel. The search starts with the character at the current INDEX location and proceeds to the end of the file or the point where a match is found. If a match is found, the channel's INDEX is set to the location of the start of the string. If no match is found, the INDEX is set to 1E9 (the value 1,000,000,000).

The string to be searched for can be up to 32 characters long and can contain don't care characters & (ampersand) i.e."T&E" will FIND "THE" as well as "TEE".

Commas and carriage returns cannot be part of FIND strings.

Numeric Output to Files

When string variables are output to a data file, the string characters and a carriage return (end of entry mark) are output exactly as would be expected. Numeric output, however, includes a leading space for positive values (or a minus sign for negative values) and a trailing space before the carriage return (end of entry mark). These spaces impart no information and merely consume file space. FLAG 11 can be used to suppress leading and trailing spaces in numeric output to files. FLAG 12 will restore normal operation of numeric output to files.

End Of File Action

Whenever the end of a file (EOF) is reached during an INPUT or PRINT statement, the running BASIC program is terminated with a system error message such as:

```
DEV A ERROR 132 IN LINE 1500
```

If some other action is preferred, the programmer may retain control within the BASIC program and implement some other action by previously executing the command

```
FLAG 5
```

In this mode of operation, when an EOF is reached, the INDEX variable for the associated channel is set to 1E9 (the value 1,000,000,000) and the INPUT or PRINT statement in progress is aborted with control returning to the succeeding statement. Some data may or may not have been transferred. An IF statement should be used after each INPUT or PRINT statement to test for the EOF condition. For example:

```
INPUT %1, D$
IF INDEX(1)>=1E9 GOTO 1000
```

The command

```
FLAG 6
```

terminates this mode of operation and results in a system error message when an EOF is reached.

Disk Error Action

Whenever a disk error occurs, the running BASIC program is terminated with a system error message such as:

```
DEV A ERROR 6 IN LINE 400
```

If some other action is preferred, the programmer may retain control within the BASIC program and implement some other action by previously executing the command

```
FLAG 9
```

In this mode of operation, when a disk error is encountered, the statement in progress is aborted and control is transferred to line 50000 in the BASIC program. At line 50000, the programmer can perform whatever special action is required including getting the error number with a

```
PEEK(10226)
```

and the line number at which the error occurred with a

```
PEEK(11774) + PEEK(11775) *256
```

The command

```
FLAG 10
```

terminates this mode of operation and results in a system error message when a disk error occurs.

Program File Manipulation Commands

LOAD file name, password (password optional)

Loads the specified program file into the BASIC work space where it may be listed, modified or run. To be LOADED, a file must be of type B (BASIC) and must have read access or the correct password must be given. If the file size exceeds the size of the work space, an OM (out of memory) error will occur.

SAVE file name, password

Saves the current contents of the work space in the specified file. If no file name, password is given, the last LOADED file is saved. To be SAVED, a file must have read/write access or the correct password must be given. If the size of the program in the BASIC work space exceeds the size of the file, an OM error will occur.

RUN file name, password, line number (password and line number optional)

Executes the specified program file by first loading it into the BASIC work space then running it. To be RUN, a file must be of type B (BASIC) and must have read or write access or the correct password must be given. Note that if a program file has write-only access, it may be RUN, but may not be LISTed and may not be LOADed or SAVEd without giving the correct password. In effect, write-only access to a BASIC program file means that the program has "RUN-only" access.

If the optional line number is specified, the program will be run starting with that line.

Money Mode Output

The money mode output format prints numerical values with two fractional digits to represent dollars and cents. The output value can also be either right justified or left justified within the print field.

Money mode output format is specified in a PRINT statement by including $L, or $R, just prior to the name of the variable (or constant) to be output in the money mode format. $L specifies left justification and $R specifies right justification.

Example:

```
X = 100
PRINT X, $R,X, $L,X
```

Prints as (_ is a space)

```
_100_ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ 100.00_ _ _ _ 100.00_ _ _ _ _ _
```

Input/Output Device Specifications

PRINT, INPUT and LIST statements may include I/O device numbers to direct I/O from or to other than the console device. I/O devices are specified by #n in such statements, where n is a constant or variable which contains the device number.

Examples:

```
PRINT #2, "X= ",X
INPUT #3, "ENTER DATE"; #1, M,D,Y
LIST #5
```

Statement Trace

To output each source line of a BASIC program as it is executed, enter the command

```
FLAG 7
```

either in the direct mode or as a program statement. Each line is output preceded by an asterisk to identify the trace function.

The trace can be stopped at any time by the command

```
FLAG 8
```

The flag commands are used to enable and disable certain system features. The form of the FLAG command is:

FLAG n

where n is one of the following:

1 Disables the Close-files-on-error feature

2 Enables the Close-files-on-error feature

3 Selects the video/keyboard as console device

4 Selects the serial console device

5 Enables user programmable disk EOF action (INDEX set to 1E9)

6 Enables program abort and system error message upon disk EOF

7 Enables BASIC statement trace

8 Disables BASIC statement trace

9 Enables user programmable disk error action (goes to line 50000)

10 Enables program abort and system error message upon disk error

11 Enables space suppression in numeric output to files

12 Disables space suppression in numeric output to files

13 Enables INPUT%n, command file operation

14 Disables INPUT%n, command file operation

19 Disable disk checksum error checking

20 Enable disk checksum error checking

21 Disable input escape on carriage return

22 Enable input escape on carriage return

OS-65U Utility Programs

A number of utility programs are provided as part of OS-65U for use in creating new files, copying files, etc.

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

Create File Utility

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

```
RUN"CREATE"
```

The program output and expected type of response are shown below:

```
CREATE FILE UTILITY
FILENAME (SIX CHARACTERS MAXIMUM)?
```

Enter a one to six character filename that is not a duplicate of an existing filename.

```
MAXIMUM LENGTH IN BYTES(DECIMAL)?
```

Enter the maximum length for the file. This number will be increased by 16 and rounded up to the next even page (256 bytes).

```
FILE TYPE
DATA = (D)
BASIC = (B)
OTHER = (O)
```

Enter D, B or O to specify the type of file. Only BASIC files can be loaded, saved and run.

```
ACCESS RIGHTS
NONE = (N)
READ = (R)
WRITE = (W)
R/W = (RW)
```

Enter N, R or W to specify the access permission for users who do not specify the correct password. Users specifying the correct password have read/write access. If write-only access is defined for a BASIC program file, it will be RUNable only and cannot be LOADed or LISTed.

FOUR LETTER PASSWORD?

Enter a four letter password or only a period (.) if the file is to have no password. If RW was the response to file type, password is not requested. CREATE then lists the responses and gives the user a chance to correct them. The file will be created and entered into the directory.

The number of free bytes remaining on disk is reported before the Create File program terminates; for example:

148736 BYTES FREE ON DISK

WARNING:

Do not allocate more memory for BASIC program files than the computer system has. Such files can be SAVEd but not LOADed or RUN because an OM error will be reported. On a 32K machine specify 8175 bytes maximum. On 48K they can be 24560 bytes maximum.

Delete File Utility

Delete removes a file entry from the directory and prepares the file's disk space for the repack program. It does not actually free up any disk space. If one or more deleted files are on a disk, the Directory Program (DIR) will report XXXX bytes recoverable indicating that the Repack Program could free up this space if executed. Deletions are performed by running the Delete Program which simply asks FILENAME and PASSWORD.

Disk Pack Utility

This utility program removes all deleted files from the system and packs all remaining files together at the start of disk memory. This places all free space at the end of memory, thus making it available for additional new files.

To pack a disk memory type:

```
RUN"PACKER"
```

The program output and expected type of response are as follows:

```
OS-65U PACKER UTILITY
PASSWORD?
```

Enter the appropriate password.

The program then begins packing the disk memory. As the packing operation proceeds, a new file directory is output showing the resultant file content and file starting addresses.

For example,

NAME	TYPE	ACCESS	ADDRESS	LENGTH
DIREC*	OTHER	NONE	26112	1024
BEXEC*	BASIC	R/W	27136	1024
CREATE	BASIC	READ	28160	8192

.

.

.

etc.

149568 BYTES FREE

14 FILES DEFINED OF 63 POSSIBLE

The packing operation can be lengthy depending upon disk file complexity and should only be executed as necessary.

WARNING: If a disk error occurs during the packing operation, the Disk Pack Utility will terminate and an error message will be printed. Most of the files on the disk being packed when an error occurs will be lost. For this reason, if the possibility of a disk error exists, a disk memory should be backed up before it is packed. (See Disk Copy Utility.)

Disk Copy Utility

This utility program is used to initialize disks and to copy the complete systems portion or files portion of a disk to another disk. To initialize or copy a disk, type:

```
RUN"COPIER"
```

The program output and expected type of response are as follows:

```
DISK COPY UTILITY

INITIALIZE  (I)
COPY SYSTEM (S)
COPY FILES  (F)
EXIT        (X) ?
```

Enter the letter I, S, F to specify the function to be performed. If X is entered, the program terminates. Continue as described in the appropriately titled section below.

Initializing

```
UNIT ?
```

Enter the appropriate alphabetic letter to specify the disk drive unit to be initialized.

A,B,C and D refer to floppy disk drives. A single drive system has unit A. A dual drive system has units A and B, etc.

E,F,G and H refer to hard disk drives. A single drive system has Unit E. A dual drive system has units E and F, etc.

If a hard disk unit is specified, a password is required.

PASSWORD ?

Enter the appropriate password.

With either type of disk, the sequence continues with:

FROM ADDRESS ?

Enter the initial disk address of the section to be initialized.

TO ADDRESS ?

Enter the final disk address of the section to be initialized.

Floppy disk addresses are 0 through 275967.

Hard disk addresses are 0 through 72898559.

The system portion occupies addresses 0 through 25087 and the files portion occupies addresses 25088 and up.

Since disk initialization is block oriented, the entered initial address is truncated and the final address is rounded up to the nearest block boundary. Each block on a floppy disk is 3584 bytes, and on a hard disk is 215040 bytes. If the entered initial and final addresses are not at exact block boundaries, a confirmation request is output. For example,

WILL INITIALIZE 0 THROUGH 275967

ALRIGHT (Y OR N) ?

Enter Y if the specified address range is acceptable or, if it is not, enter N and re-enter the addresses.

Disk initialization will take place at this time. Due to the possible length of time involved in hard disk initialization (up to 85 minutes), the calculated duration is output, for example,

WAIT 7.5 MINUTES

FROM UNIT ?

Enter the appropriate unit letter to specify the unit from which data is to be copied.

If a hard disk unit is specified, an address range for the copy must also be specified:

FROM ADDRESS ?

TO ADDRESS ?

Enter the appropriate initial and final disk address of the section from which to copy.

Since disk copying is block oriented, the entered initial address is truncated and the final address is rounded up to the nearest block boundary. Each copyable block on a hard disk is 3584 bytes. If the entered initial and final addresses are not at exact block boundaries, a confirmation request is output. For example,

WILL COPY FROM 0 THROUGH 25087

ALRIGHT (Y OR N) ?

Enter Y if the specified address range is acceptable or, if it is not, enter N and re-enter the addresses.

The copy sequence continues with:

TO UNIT ?

Enter the appropriate unit letter to specify the unit to which data is to be copied. OS-65U systems can be copied to floppy disks only in Versions 1.0 and higher.

If a hard disk unit is specified, an initial address for the copy must also be specified:

```
TO ADDRESS ?
```

Enter the appropriate initial disk address of the section to which data will be copied. Hard disk systems must begin on disk cylinder boundaries; therefore, the entered address is truncated to the next lower cylinder boundary. Each cylinder is 215040 bytes. If the entered address is not exactly on a cylinder boundary, a confirmation request is output. For example,

```
WILL COPY TO Ø
ALRIGHT (Y OR N) ?
```

Enter Y if the specified address range is acceptable or, if it is not, enter N and re-enter the address.

Rename File Utility

This utility program is used to change the name and password of an existing file.

To rename a file, select the unit on which the file resides by typing

DEV"u"

where us is the appropriate unit designation. Then type:

RUN"RENAME"

The program output and expected type of response are as follows:

RENAME FILE UTILITY

FILENAME?

Enter the name of the file to be renamed.

PASSWORD?

Enter the password of the file to be renamed or a period (.) if the file has no password.

The program will locate the specified file in the disk directory. If the name and password do not match those of any existing file, the message FILE NOT FOUND is output and the program terminates. When the specified file is found, the program continues.

NEW FILENAME?

Enter the new name for the file.

NEW PASSWORD?

Enter the new password for the file or a period (.) if the file is to have no password.

The filename and password are changed and the following message is output:

RENAMED: old filename, password AS: new filename, password

File Print Utility

This utility program is used to print selected sequential entries from a sequential file or selected fields from a randomly addressable file.

To print a file, type:

RUN"FPRINT"

The program output and expected type of response are as follows:

FILE PRINT UTILITY

UNIT?

Enter the unit designation for the file to be printed.

CONSOLE (C) OR PRINTER (P) OUTPUT?

Enter C or P to specify the output device.

FILENAME?

Enter the name of the file to be printed.

PASSWORD:

Enter the appropriate password or a period (.) for no password.

SEQUENTIAL (S) OR RANDOM (R) TYPE?

Enter S or R to specify the type of data file access to be used.

If a random type file has been specified, skip the discussion for sequential access which follows.

STARTING INDEX?

Enter a value for the starting file index from Ø through the total file size minus 1.

FINISHING INDEX (Ø OUTPUTS TO END OF FILE)?

Enter a value for the last file index for the print. This value must be larger than the starting index and less than the total file size. To print to the end of file, merely enter Ø.

The sequential print format is as shown in the following sample print:

```
SEQUENTIAL FILE PRINT FROM INDEX Ø TO 64

FILE:     PHODIR

INDEX:    CONTENTS

   Ø      GORDON KNOTTS

  14      566-1955

  23      HARRY BONNELL

  37      789-3245

  46      CHARLES EVANS

  60      324-7073

  69
```

For a random type file, the sequence continues as follows:

RECORD LENGTH?

Enter the length of records in the file. (All records in a randomly addressable file are the same length.)

OFFSET TO FIRST RECORD?

Enter Ø for no offset or a positive offset value if the first file record begins at an index value other than zero.

STARTING RECORD NUMBER?

Enter the number of the first record to be printed. The first record in the file is record number zero.

FINISHING RECORD NUMBER (Ø OUTPUTS TO END OF FILE)?

Enter the number of the last record to be printed or Ø to output all records to the end of file.

NUMBER OF FIELDS TO BE PRINTED?

Enter the number of fields within each record that are to be printed. Any number of fields within each record can be printed.

FIELD 1 LOCATION?

Enter the relative location within the record of the first field to be printed. If it is the first field within the record, its relative location is Ø. This message is repeated to permit entry of the location of each of the fields to be printed.

The random dump format is as shown in the following sample print:

```
RANDOM FILE PRINT FROM RECORD Ø TO 2
FILE:   PHODIR
RECORD  Ø
FIELD   CONTENTS
  1     GORDON KNOTTS
  2     566-1955
RECORD  1
FIELD   CONTENTS
  1     HARRY BONNELL
  2     789-3245
RECORD  2
FIELD   CONTENTS
  1     CHARLES EVANS
  2     324-7073
```

Copy File Utility

This utility program is used to copy the contents of any file to another file.

To copy a file type:

RUN"COPYFI"

The program output and expected type of response are as follows:

FROM

UNIT?

Enter the unit designation for the source file.

FILENAME?

Enter the name of the source file.

PASSWORD?

Enter the password of the source file or a period (.) if the file has no password. The specified file will be located in the appropriate disk directory. If the name and password do not match those of any existing file, the message FILE NOT FOUND is output and the program terminates. When the specified file is found, the program continues.

TO

UNIT?

FILENAME?

PASSWORD?

Enter the appropriate unit designation, name and password of the target file in response to each of the above questions. The target file will be located in the appropriate disk directory, the copy will be performed and the COPYFI program will terminate.

File Dump Utility

This utility program is used to dump the total contents of a data file.

To dump a file, type:

RUN"FDUMP"

The program output and expected type of response are as follows:

FILE DUMP UTILITY

UNIT?

Enter the unit designation for the file to be dumped.

TERMINAL WIDTH?

Enter 1 to 64 to specify the number of characters to be output on each line.

CONSOLE (C) OR PRINTER (P)?

Enter C or P to specify the output device.

FILENAME?

Enter the name of the file to be dumped.

PASSWORD?

Enter the appropriate password or a period (.) for no password.

FILE OFFSET?

If the dump is to start at the first character of the file, enter Ø, otherwise enter a positive offset value.

NUMBER OF BYTES TO BE DUMPED (Ø=ALL)?

Enter the number of bytes to be dumped or Ø if the whole file is to be dumped.

The dump format is as shown in the following sample dump:

```
DUMP OF FILE:  FDUMP


CHARACTER SUBSTITUTION OF KEY:
NULL                = _
CARRIAGE RETURN     = #
OTHER CONTROL CHAR = @


 0    _@P@_________@@____# 1 CLOSE# 5
32     DIMD$(80)# 10 INPUT "FILENAME"
64
```

Disk Change Utility

This utility program is used to display the contents of any addressable disk memory location in decimal or hexidecimal and as an ASCII character and to change the contents of any such location.

THIS PROGRAM MUST BE USED ONLY WITH EXTREME CAUTION BY EXPERIENCED PERSONNEL OR CATASTROPHIC SYSTEM FAILURES MAY RESULT.

To change disc memory, type:

RUN"CHANGE", password

The program output and expected type of response are as follows:

DISK CHANGE UTILITY

MODE: HEX(H), DEC(D)?

Enter H or D to specify hexidecimal or decimal mode for address and data input/output. Enter X to terminate the program.

UNIT?

Enter the unit designation for the unit on which changes are to be made. Enter X to terminate the program.

ADDRESS OFFSET?

Enter the offset value to be added to specified address values to arrive at the absolute disk address to be displayed/changed.

ADDRESS?

Enter an absolute disk address or a relative disk address if a non-zero address offset was specified. Enter a period (.) to return to the ADDRESS OFFSET question.

Use of the address offset value permits easier relative references to disk memory portions of the system or files. For example, the difference between disk and ram addresses for the system is 3072 (hex CØØ). Entry of this value for an address offset permits reference to system locations on disk by the corresponding ram addresses. Then, for example, to change the memory size locations (11702 and 11703 in ram, 14774 and 14776 on disk) merely enter the ram address 11702. The offset value 3072 is automatically added internally by the program giving 11702 + 3072 = 14774, the proper disk address.

The contents of disk memory locations are displayed as follows:

aaaaaaaa c dd?

where the a's represent the address, c represents the ASCII character represented by the data byte and dd represents the numerical value of the data byte.

Enter the new value for the data byte if it is to be changed.

Enter a slash (/) to display the contents of the next consecutive address without changing memory contents.

Enter a back slash ( ) to display the contents of the previous address without changing memory contents.

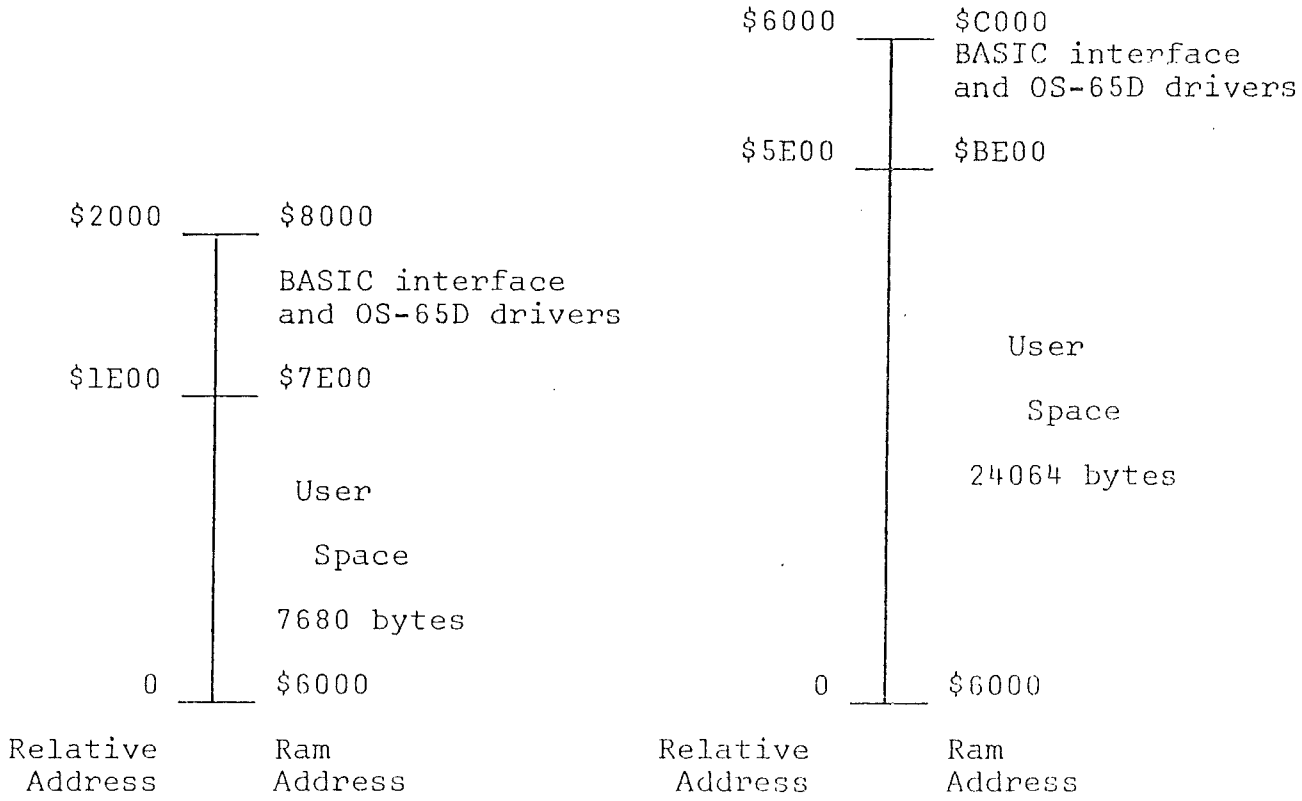
Enter a period (.) to return to the ADDRESS question. Enter an X to terminate the program.

When all needed disk changes have been made, the CHANGE program must be terminated by typing an X or the changes may not be effected.

Note that all numerical output by this program is in the number base specified in response to the MODE question. Likewise, all input must be expressed in the same number base. Entry of a number in the wrong base will result in a repetition of the last program output if the error is detectable. However, not all such errors are detectable so user caution is advised.

Machine Code Loader Utilities

These utility programs are used to create BASIC programs which also contain machine (assembly) code within their program area. In such programs, the machine code resides in the lower portion and the BASIC program occupies the uppermost portion of the program area. LOAD32 is used in 32K ram systems and LOAD48 is for use in 48K ram systems. These programs are structured as shown here:



Any portion or all of the user space in each program can be loaded with machine code routines developed on an OS-65D or WP-1A system.

Then the user's BASIC program which interfaces to the machine code routines can be entered in the remaining space above the machine code.

The procedure for constructing such a program follows.

Create a program file for the program using the CREATE utility. Define its size large enough to contain both the needed machine language routines and the BASIC program which interfaces with these routines.

Develop the machine language routines on an OS-65D or WP-1A system. These routines must be assembled to run a selected ram address within the LOAD32 or LOAD48 user space. After the routines are developed, save the resulting object code on the 65D or WP-1A diskette.

Next, boot the OS-65U system and type:

RUN"LOAD32 or RUN"LOAD48, as appropriate.

The program output and expected type of response are as follows:

```
MACHINE CODE LOADER UTILITY - XXK
TO RETURN TO OS-65U TYPE:  GXXXX
A*
```

The program has now entered an OS-65D disk operating system within its program area and will respond to any OS-65D commands. At this time, the 65D or WP-1A diskette containing the object code for the machine language routines developed earlier can be inserted into the disk drive. The object code can then be loaded into the user area with the C (call) command. For example, to load object code from track 50, sector 1 into the user area at hex 6000 type:

```
C6000=50,1
```

(Refer to the OS-65D manual for further information on its operation.)

When the needed machine code routines have been loaded into the user space, return to OS-65U by reinstalling the 65U diskette and typing:

GXXXX where XXXX is the address output earlier.

OS-65U will type:

OK

At this time, define the size of the machine language portion of the new program and clear the BASIC portion by typing:

NEW XXXX

where XXXX is the decimal size (number of bytes) of the machine language portion.

The BASIC program may now be entered and the resulting combined program saved on disk by typing:

SAVE filename, password

where the filename and password are as defined when the original program file was CREATEd in the first step.

Peripheral Device Numbers

Input	Output		Device Flag Bit*
1		Serial console device (ACIA)	0
	1	Serial console device	0
2		Keyboard**	1
	2	Video display (440 or 540)**	1
3		UART input (430)	2
	3	UART output (430)	2
4		Memory input	3
	4	Memory output	3
	5	Line printer	4
8		CA-10X 16 port input	7
	8	CA-10X 16 port output	7

*To activate a specific device, set this bit in the appropriate Active Device Flag Byte. For example, to output to the line printer and serial console, use

```
POKE 11686,17
```

or its equivalent

```
POKE 11686, 2^4+2^0
```

where 4 and 1 are the respective peripheral device bits.

**Versions of OS-65U with a P after the version number support the Polled Keyboard as device 1 and only the 540 video display as device 2. The Polled Keyboard PROM is required at $FDØØ to support the polled keyboard.

Console Device Control Characters

The console device - either a serial terminal, device 1, or a 440 or 540 video terminal, device 2 - provides a number of control character commands for controlling output to the console and BASIC program execution:

(control) C - stops BASIC program listing or execution at the end of the current statement

(control) S - stops all output pending input of a (control) Q

(control) Q - restarts output that was stopped by (control) S or D

(control) O - causes output to be "thrown away" pending input of another (control) O

(control) D - limits output to one "page" (video screen) at a time, then stops pending input of a (control) Q which results in the next "page" being displayed

(control) W - terminates paging of output that was initiated by a (control) D

Line Printer Page Control

The line printer, device 5, includes a page control feature that skips six lines every 66 lines in order to provide a top and bottom margin on each page of printer output. When the system is booted, the first print line position is assumed to be under the print head. If it is not, the printer paper should be so adjusted. Proper page control will be maintained automatically thereafter as long as the paper position is not manually changed.

The number of printed lines per page (page size - usually 66 - minus top and bottom margins) can be changed by POKEing the required number into location 14457. If the number 66 is POKEd, paging is effectively disabled. If necessary, the total number of lines per page can be changed by POKEing 14387.

The following sample statement may be used to cause a slew to top-of-form:

```
100 PRINT #5: IF PEEK (15908)<>PEEK (14457) GO TO 100
```

The page control feature is useful on printers that do not have such a feature internally such as the Centronics 779. Users of printers that do have a paging feature, such as the Okidata 22, may prefer to disable the feature in OS-65U by a POKE 14457,66.

Indirect Files

Indirect files provide a mechanism for merging programs and for transferring BASIC programs to or from OS-65D systems and OS-65U systems. Ohio Scientific's Small Systems Journal, Volume 1, No. 5, contains a detailed discussion of indirect files under OS-65D.

Prior to using indirect files under OS-65U, the following steps must be taken:

1). Preset a constant memory size into OS-65U that leaves sufficient space at the top of ram memory to hold the indirect ASCII file. Use the CHANGE utility as follows (user input is underlined):

RUN"CHANGE"

DISK CHANGE UTILITY

MODE: HEX(H), DEC(D) ? H

UNIT? A (or other appropriate disk unit designation)

ADDRESS OFFSET? CØØ

ADDRESS? 2DB6

ØØØØ2DB6 ØØ? 8Ø (low byte of first free ram address)

ØØØØ2DB7 ØØ? 76 (high byte of first free ram address)

ØØØØ2DB8 A9? X

OK

In the above example, the last location left for OS-65U BASIC programs is 767f and the first location available for the indirect file is 7680.

The OS-65U system must be rebooted to effect the change in memory size.

2). Enable the indirect file function with the following BASIC statements:

```
POKE 14646,91
POKE 14721,24
```

Now, for example, a BASIC program, after being LOADed into the work space, can be output to the indirect file by typing:

```
LIST[ (return)
(program listing)
]]@
```

The same, or another program previously output to the indirect file, can be input to the work space by typing:

```
(control) X (return)
```

0079 BEGINNING OF PROGRAM START
007B START OF VARIABLES POINTER
007D ARRAY POINTER
0034 HIGHEST MEMORY LOCATION IN BASIC
0036

OS-65U

Memory Locations

0086 CURRENT LINE # (LO HI)

HEX	DECIMAL	
2D89	11657	Memory input pointer (2 bytes; low, high)
2D8D	11661	Memory output pointer (2 bytes; low, high)
2D90	11664	Console input device number
2D91	11665	Console output device number
2D92	11666	Memory I/O pointer for indirect files (2 bytes; low, high)
2D94	11668	Active input device flag byte
2DA6	11686	Active output device flag byte
2DB6	11702	Constant memory size (Ø=use all available memory) (2 bytes; low, high) system must be rebooted to effect change.
6000	24576	Start of BASIC program work space
4D56	19798	CA-10X 16-port interface device index (Ø,2,4,...,3Ø correspond to port 1,2,3,...,16.255 programs interface to scan all input ports, output to port 1.)
2E1A	11802	Number of floppy disk drives on system
340A	13322	Number of CD-74 disk drives on system
ØB6E	1390	BASIC line delete character (Ø=64)
Ø572	1394	BASIC character delete character (←=95)
0566	1382	SCRAPE ASCII BELOW (20)
056A	1386	SCRAPE ASCII ABOVE (7D)

2320 OOTCH

INTERRUPTS	HEX	DEC
NMI	0130	304
IRQ	FFAO	65440
BRK	01C0	448

Page 0 is configured as follows:

KB-6 uses all locations 0-D4 and FF

KB-9 uses all locations 0-DC and FF

from KIM release

STD MICROSOFT BASIC (6 DIGIT) ZERO PAGE ALLOCATION

KB-6	KB-9	DESCRIPTION
0	0	JMP to WARM START BASIC.
6	6	Address of routine to transfer USR argument to y,A (AYINT).
8	8	Address of routine to transfer (y,A) to result of USR function (GIVAYF).
13	14	FLAG set to FF if output is suppressed (CONTROL O mode). Set to 0 otherwise.
14	15	Number of NULLS to print.
15	16	Current terminal column (equal to POS (0)).
16	17	Line Length.
17	18	Position beyond which there are no more comma fields. Equal to 14*(INT (line length/14)-1).
1A	1B	Input buffer .72 decimal bytes.
76	78	Pointer to start of program.
78	7A	Pointer to start of simple variable table.
7A	7C	Pointer to start of array table.
7C	7E	First location unused by array table.
7E	80	Lowest location used by string data.
82	84	Highest memory location in use by BASIC.
84	86	Current line number.
A9	AE	Floating accumulator
B9	CD	Routine to read a character from current program position.
D1	D8	Current random number.
D5	DD	First unused page 0 location.
FF	FF	Used by STR$ function.

I/O Device Base Addresses

PAGE	TYPE	START ADDRESS	FUNCTION
0	PIA	$E810	Keyboard
1	PIA	$E820	IEEE-488
2	VIA	$E840	USR PORT cassette

Location not specified are used but have no clear one function definition.

PET PAGE ZERO MEMORY MAP

FROM	TO	DESCRIPTION
000	--	$4C constant (6502 JMP instruction).
001	002	USR function address lo, hi.
Terminal I/O maintenance		
003	--	Active I/O channel #.
004	--	Nulls to print for CRLF (unused).
005	--	Column BASIC is printing next.
006	--	Terminal width (unused).
007	--	Limit for scanning source columns (unused).
008	--	Line number storage before buffer.
009	--	$2C constant (special comma for INPUT process).
010	089	BASIC INPUT buffer (80 bytes).
090	--	General counter for BASIC.
091	--	$00 used as delimeter.
092	--	General counter for BASIC.
Evaluation of variables		
093	--	Flag to remember dimensioned variables.
094	--	Flag for variable type; 0#numeric; 1 ÷ string.
095	--	Flag for integer tape.
096	--	Flag to crunch reserved words (protects '& remark).
097	--	Flag which allows subscripts in syntax.
098	--	Flags INPUT or READ.
099	--	Flag sign of TAN.
100	--	Flag to suppress OUTPUT (+ normal; − suppressed).
101	--	Index to next available descriptor.
102	103	Pointer to last string temporary lo; hi.
104	111	Table of double byte descriptors which point to vaiables.
112	113	Indirect index #1 lo; hi.
114	115	Indirect index #2 lo; hi.
116	121	Pseudo register for function operands.
Data storage maintenance		
122	123	Pointer to start of BASIC text area lo; hi byte.
124	125	Pointer to start of variables lo; hi byte.
126	127	Pointer to array table lo; hi byte.
128	129	Pointer to end of variables lo; hi byte.
130	131	Pointer to start of strings lo; hi byte.
132	133	Pointer to top string space lo; hi byte.
134	135	Highest RAM adr lo; hi byte.
136	137	Current line being executed. A zero in 136 means statement executed in a direct command.
138	139	Line # for continue command lo; hi.
140	141	Pointer to next STMNT to execute lo; hi.
142	143	Data line # for errors lo; hi.
144	145	Data statement pointer lo; hi.

Expression evaluation		
146	147	Source of INPUT lo; hi.
148	149	Current variable name.
150	151	Pointer to variable in memory lo; hi.
152	153	Pointer to variable referred to in current FOR-NEXT.
154	155	Pointer to current operator in table lo, hi.
156	--	Special mask for current operator.
157	158	Pointer to function definition lo; hi.
159	160	Pointer to a string description lo; hi.
161	--	Length of a string of above string.
162	--	Constant used by garbage collect routine.
163	--	$4C constant (6502 JMP inst).
164	165	Vector for function dispatch lo; hi.
166	171	Floating accumulator #3.
172	173	Block transfer pointer #1 lo; hi.
174	175	Block transfer pointer #2 lo; hi.
176	181	Floating accumulator #1. (USR function evaluated here).
182	--	Duplicate copy of sign of mantissa of FAC #1.
183	--	Counter for # of bits fo shift to normalize FAC # 1.
184	189	Floating accumulator #2.
190	--	Overflow byte for floating argument.
191	--	Duplicate copy of sign of mantissa.
192	193	Pointer to ASCII rep of FAC in conversion routine lo; hi.
RAM subroutines		
194	199	CHRGOT RAM code. Gets next character from BASIC text.
200	--	CHRGOT RAM code regets current characters.
201	202	Pointer to source text lo; hi.
203	223	Next random number in storage.
OS page zero storage		
224	225	Pointer to start of line of cursor loc lo; hi.
226	--	Column position of cursor.
227	228	General purpose start address indirect lo; hi.
229	233	General purpose and address direct lo; hi.
234	--	Flag for quote mode on/off.
238	--	Current file name length.
239	--	Current logical file number.
240	--	Current primary address.
241	242	Current secondary address.
243	244	Pointer to start of current tape buffer lo; hi.
245	--	Current screen line #.
246	--	Data temporary for I/O.
247	248	Pointer to start loc for O.S. lo; hi.
249	250	Pointer to current file name lo; hi.
251	254	Unused.
255	--	Overflow byte that BASIC uses when doing FAC to ASCII conversions.

Page 1

62 byte on bottom are used for error correction in tape reads. Also, buffer for ASCII when BASIC is expanding the FAC into a printable number. The rest of page 1 is used for storage of BASIC GOSUB and for NEXT context and hardware stack for the machine.

VARIABLE ALLOCATION

Space is allocated for variables only as they are encountered. It is not possible to allocate an array on the basis of 2 single elements, hence the reason to execute DIM statement before array references. Seven bytes are allocated for each simple variable whether it is a string, number, or user defined function.

The first two bytes give the name of the variable:

	byte 1	byte2
INTEGER	first chr + 128	Second chr + 128 or 128
FLOATING	first chr	second chr or 0
STRING	first chr	second chr + 128 or 128

The last five bytes give the value of a variable, or a descriptor to the rest of the data:

INTEGER

actual value				
256 * HI	LO	0	0	0

FLOATING

actual value in binary floating point				

STRING

	pointer			
chr count	LO	HI	0	0

The simple string variable points to a location in high memory, where the actual characters are stored.

Examples of declaration and storage

```
15% = 90
     201   181   0   90   0   0   0
C$ = "HELLO"
      67   128   5   .   .   0     0
```

H	E	L	L	O

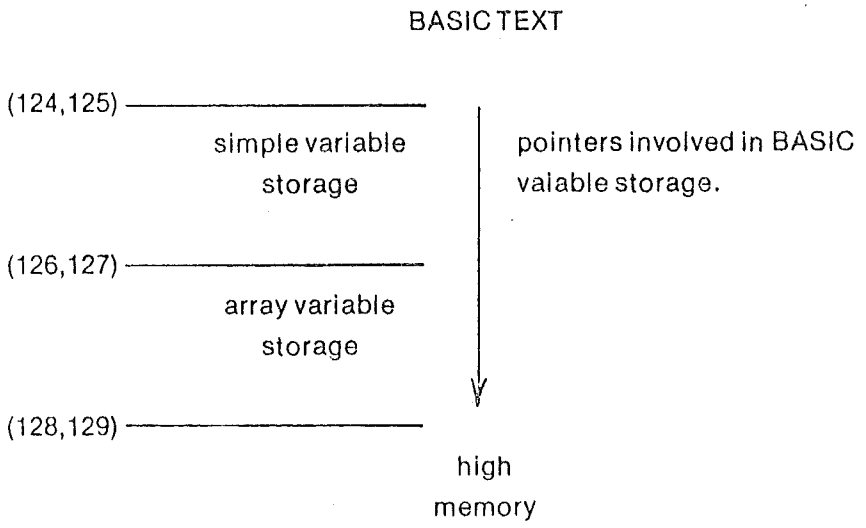
Locations 124 and 125 contain the first address of memory where a simple variable name will be found.By incrementig the address by 7 each time the ext simple variable name in the table is encountered.The end of the variables is defined-by the address in 126 and 127.

Locations 126 and 127 also define the start of array storage. The first two bytes of array descriptors arethe same as simple variables but the next five bytes are special as follows:

	byte 3	byte 4	byte 5	byte 6	byte 7
VECTOR ARRAYS	7 + (size + 1)* (dim)*A	0	1	0	size + 1

where A = 2 for Integer, = 3 for string, or = 5 for floating.

By incrementing the search address by the current byte #3 of the descriptor each time, the next array variable is reached. Locations 128 and 129 contain the ending address of this table.



Because the variables are divided in storage between arrays and simple variables insertion of an additional simple variable is a bit more complicated once an array has been defined. First, the entire array storage area must be block moved upward by seven bytes and the pointers adjusted upward +7. Finally, the simple variable can be inserted at the end of simple variable storage.

> If large arrays are defined and initialized first before simple variables are assigned, much execution time can be lost moving the arrays each time a simple variable is defined. The best strategy to followin this case is to assign a value to all known simple variables before assigning arrays. This will optimize execution speed.

Functions of NEW and CLR on data pointer:

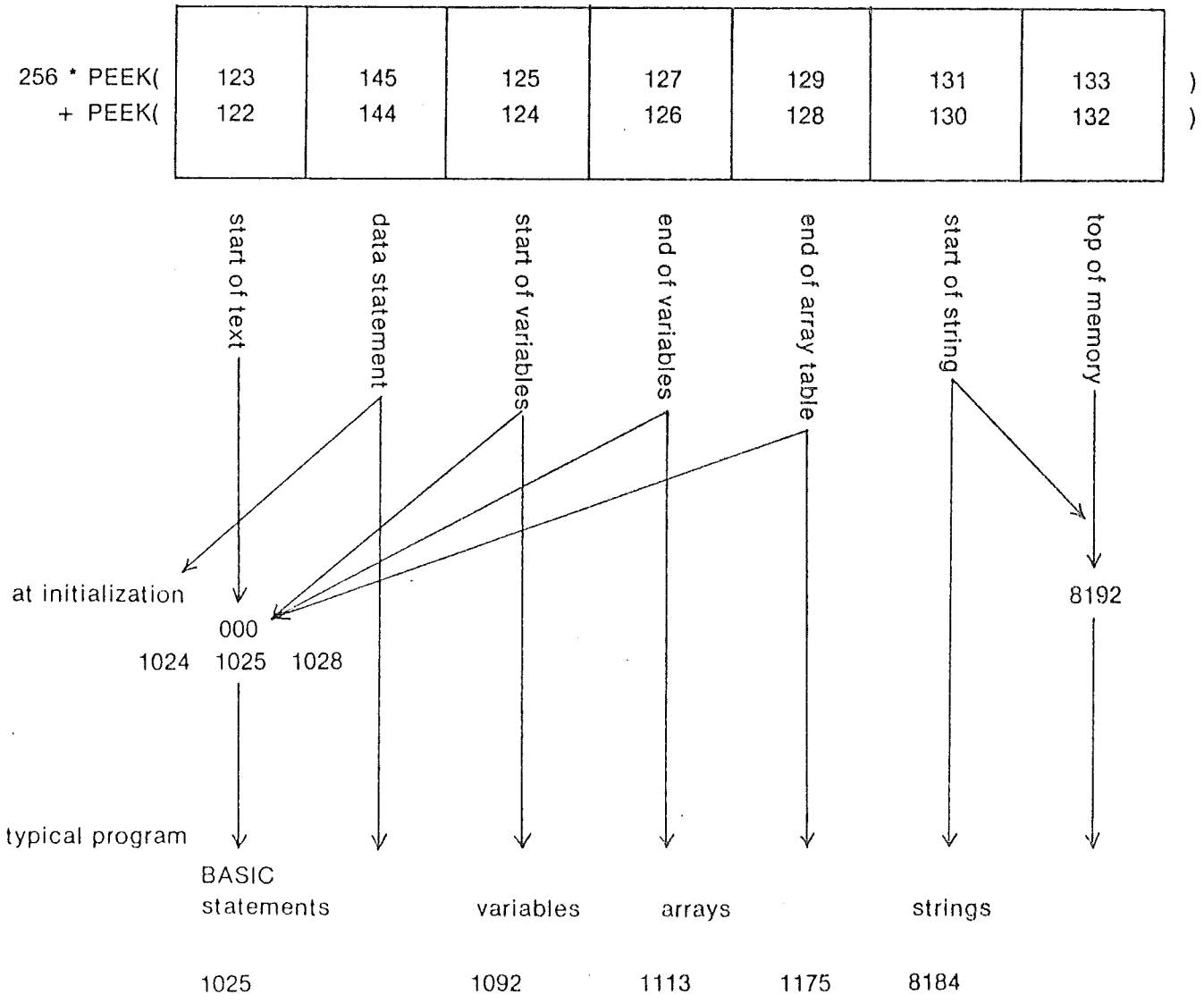
C L R

String pointer equated to top of memory data pointer to start of text – 1 end of array table to start of variables end of simple variables to start of variables.

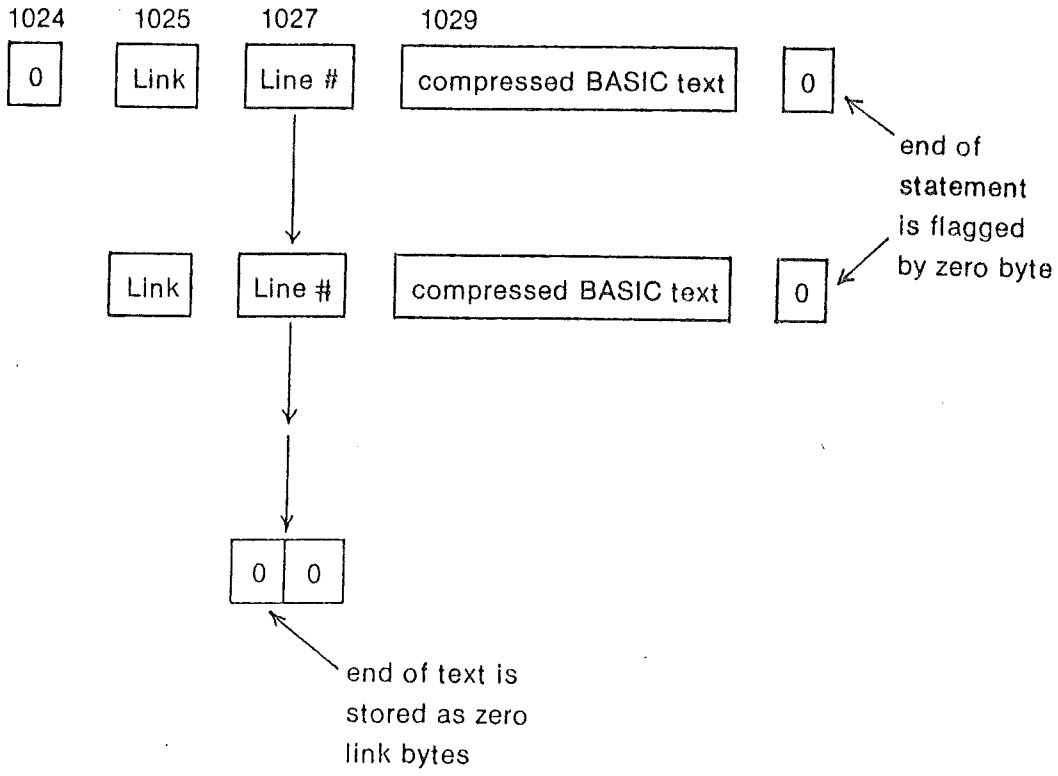
N E W

String pointer equated to top of memory data pointer to start of text – 1 end of array table to start of text +3 end of simple variables to start of text + start of variables to start of text +3.

PRINCIPAL POINTERS INTO PET RAM



HOW BASIC STATEMENTS ARE STORED



OS-65U Floppy Disk Error Codes

1	Drive not ready
2	Seek error
3	Invalid unit number
4	Can't find track zero
5	Can't find index hole
6	Diskette write protected
7	Track unsafe (can't verify write)
8	Incomplete Header
9	Header - Framing Error (FE)
10	Header - Overrun (OR)
11	Header - OV,FE
12	Header - Parity Error (PE)
13	Header - PE,FE
14	Header - PE,OV
15	Header - PE,OV,FE
16	Data Field - Incomplete
17	Data Field - Framing Error
18	Data Field - Overrun
19	Data Field - OV,FE
20	Data Field - Parity Error
21	Data Field - PE,FE
22	Data Field - PE,OV
23	Data Field - PE,OV,FE
24	Checksum Error
25	Unit Out of Service
26	OS-65U Header Found
27	Track Zero Verification Error
>76	Track Out of Range

OS-65U Hard Disk Error Codes

1	Drive Not Ready
2	Seek Error - Timeout
3	Invalid Unit Number
4	Restore Timeout
5	DMA Failed to Terminate
6	Write Protect Error
7	Sector Unsafe (Can't verify write)
8	Sector Header - Checksum Error
9	Sector Header - Cylinder Mismatch
10	Sector Header - Track Mismatch
11	Sector Header - Sector Mismatch
16	Data Field - Checksum Error
24	Status Error
25	Unit Out of Service
>82	Cylinder Out of Range

OS-65U File System Error Codes

128 - File not found

129 - Channel not open

130 - Access Right violation

131 - Executability violation

132 - End of file

133 - Channel already open

BASIC ERROR

US UN-SUBSCRIPTED VAR

SN SYNTAX

TM

OM OUT-OF-MEMORY

OV OVERFLOW

Creating A New Slave System On OS-65U

1. Initialize (boot) the CD-74 disc.
2. Select System No. 1.
3. Enter the password.

   "OK" will be output
4. List lines 50-250 and enter the new system name, password, and length in a DATA statement after the last one in this group of lines; but before line number 250, then SAVE the program.
5. Type RUN to get a new systems directory and to see the starting address of the new system. Select System No. 1 again.
6. RUN"COPIER and copy system and files to the new system address. (If this portion of the disk hasn't been initialized, that must be done before the copy.)
7. Type RUN"SYSDIR and select the new system.
8. When it comes up LOAD"CREATE and LIST 4, change this line to HS= size of this system in bytes as printed in the system directory. Then SAVE this program.
9. Do the same with DIR, line 60, as was done with CREATE in Step 8.
10. Do the same with PACKER, line 50, as was done with CREATE in Step 8.

WARNING: Steps 8, 9 and 10 are not merely for documentation purposes. The new slave system as well as any other systems on the disk may be destroyed if these steps are not properly performed.

Guide to OS-65U Directory.

The OS-65U diskette contains several demonstration programs and data files. DIREC* is the actual directory and should not be tampered with. It is supported by the CREATE program, the DIRECTORY program and the OPEN, LOAD, RUN and SAVE routines in BASIC.

BEXEC* is the initialization program which is automatically executed whenever the machine is reset. The user can modify the end of BEXEC* as desired to customize the system. For instance, a menu can be added to select programs or have BEXEC automatically print a directory.

BUS 1, BUS 2, and BUS 3 are three short programs that could be used as a basis of a small business applications program system. These three programs utilize two data files - ACC1 and ACC2. Consult the included listings in conjunction with this discussion. BUS 1 maintains a random access file named ACC1 which contains account numbers and a current balance of each account. ACC1 has been pre-loaded with dummy data on the diskette. It has Accounts 0 thru 200 and each account, typically, has an amount of money equal to twice the account number in it. Let's examine the structure of the program to get a better insight into the operation of OS-65U. Line 10 opens ACC1. The program then asks for the account number on line 20. Since it was predetermined that the account ranges would be between 0 and 199, line 30 protects against a random access out of range. The account number is entered and used to calculate the index of ACC1 in line 35. Line 40 has conditional input for checking or modifying the program. Let's consider the case of "checking" where the program falls through to line 70. It simply inputs the account number which is variable A and the current value as variable B. It then prints the account and the amount using right money mode to force numeric output. If "modify" shows in line 40, the program falls through to line 100 in which it asks for the new amount variable Y, it then reprints out account number and the amount of the random file. The format of the random file is two entries sequentially loaded and accessed within random address organization of 20 characters. The total length of account number and the amount must be less than 20 characters including the carriage return at the end of each entry. Each new account entry starts exactly a multiple of 20 characters from the beginning of the file so that this file uses both random access for fast access and easy editing and then within that random accessed field, it sequentially addresses two entries for ease of programming.

BUS2 is another program which simply maintains a pure sequential file with an added twist. The file maintains account number and description of the account, such as, company name or part descripton. The file that this operates on is ACC2. This file, although sequential, uses a trick to quickly find the end of the file. A simple approach to finding the end of the file would be to physically enter a character field as end of file such as the word "END". The user would have to compare each string input for a match with the word "END" to find the end of file, a tedious process. Furthermore, when upon ending the file, it would be necessary to write over the word "END" and physically rewrite the word "END". This file uses an alternate approach. It places a pointer to the end of the file at the beginning of the file and allocates up to 10 characters for this end of file indicator. The end of file indicator is first loaded in a totally new ACC2 file by the use of the NEW command in line 20 where the program falls through line 500. Here, if the user types in the word PASSWORD, it loads the variable Z into the file at location 0 so that a new file has one entry at location 0 which is the number 10. The 10 is used by the program as the first place in file where actual data can be placed. Once this pointer to the end of the file is placed with the NEW command, the ADD command can be used to add additional account descriptions to the file. By the way, the file ACC2 currently has several entries in it. Let's consider a pass through the program for an ADD command. The file is opened, the end of file pointer is loaded in line 40 and stuffed into variable Z. The index to the file is set to the beginning of the actual data which has been pre-specified to be 10, then jump to line 300 where the index is now set to point to the physical end of the existing file, input an account number and description, print these to the file and update the variable Z with the new index. If another entry is desired, go back to line 310, add another account and description and update the index to the file. When all additional entries have been added to the file, the variable Z which is the new physical end of file pointer is stuffed into location 0 in the file by lines 400 and 410. The file is then closed and the program exits. To list the files, open the file, input the end of file pointer in line 40, variable Z, then set the index to the file to location 10 in the file. Input account numbers and descriptions successively and print them out on a terminal until the index to the file equals or is greater than the end of file pointer that was at the beginning of the file.

Then, terminate the program. ACC1 and ACC2 and BUS 1 and BUS 2 demonstrate two fundamental programming techniques which are very useful for small business programming. A very efficient small business package can be generated by utilizing random access numeric files for the data that is frequently changed in a small business such as the payable accounts, receivable accounts and inventory. Sequential files can then be utilized to generate reports with descriptors so that a typical accounts receivable report would utilize account descriptions in a sequential file in conjunction with numeric data in a random access file. An example of how to construct such a report is shown in BUS 3. Let's examine the source of listing for BUS 3.

BUS 3 prints out the accounts and descriptions specified by ACC2 along with the physical balance maintained for those files in ACC1. No arithmetic operations are actually performed on the balance but this would be a very straight forward extension of the program. First, ACC1 and ACC2 are opened, then the program inputs the end of file pointer from ACC2, sets the index for Channel 2 (which corresponds to ACC2) equal to the beginning of data and inputs an account as a descriptor. Then it uses that account number to set the index for random access in ACC1. BUS3 inputs the account number from ACC1 and just to be extra safe, which never hurts in business programming, compares the account pulled from ACC2 with the one pulled from ACC1. If they are not equal,it specifies that some sort of error occured and exits the program. If they are equal,it pulls a physical balance from ACC1, prints the account number, description and the current balance and via line 120, continues this process until it hits the end of file pointer in ACC2. The use of redundant fail safe comparisons and messages is extremely important in the small business programming to constantly check the user, the programs, and the system. It certainly doesn't hurt to have redundant tests such as the comparison in line 90 of this program.

PHONE and PHODIR are a program and data file for a phone directory which seems to be the standard way of demonstrating BASIC file capability.The phone directory program here is obviously much shorter than phone directory programs in OS-65D, of course, due to the fact that all the major file operations are handled via BASIC in OS-65U.

The phone directory program creates and maintains a simple sequential telephone directory file which utilizes an end file pointer at the beginning of the program. Its operation is very similar to BUS 2. The program has four commands - NEW, FIND, ADD and EXIT. The NEW command places the end of file pointer at the beginning of the file. ADD then utilizes this pointer to add additional names and phone numbers to the file. EXIT simply exits the program and closes the file on its way out insuring that the most recent data is stored in the file. FIND command however makes use of the very powerful FIND command in BASIC. This is accomplished through line 70 and 80. The user inputs the name under the FIND command and then line 80 uses the FIND command in BASIC to find the INDEX of the beginning of that entry in the file. The FIND command automatically reports an index of one to the ninth power or 1E9 if it does not find the string in the file specified. Therefore, line 90 immediately checks INDEX to be greater than 1E8 to check if the entry is not found. An important feature of the FIND command that can get programmers in trouble is that the FIND command starts from the specified index within the file, that is, it searches from specified starting index of the file to the end of the file. It is therefore important that the user be sure that the index for a particular channel or file is set to 0 before executing the FIND command if he desires to search the entire file. Note that in line 45, the INDEX of Channel 1 is set equal to 0 each time through the command loop to insure that the FIND command always searches from the beginning of the file. The user can use the LISTER program to list the PHODIR file. He simply runs LISTER and types in PHODIR with response to file name in which case the program will list out the file.

LABLE is a powerful mailing label program. This program could be used directly as a general purpose mailing label printer program by simply adding the appropriate formatted PRINT output for a particular line printer. It is set up to be used with any data file as specified by the user when operated. The system also includes sample label file TLABLE which includes some Ohio Scientific dealers. The program has five functions - LIST, ADD, EDIT, NEW and EXIT. The mailing program utilizes an extremely powerful indexed randomly addressable file. This file system allows indexed addressing, random access and sequential access all on one file to provide virtually instantenous response to any command no matter what size the actual mailing label file is. The file structure and operation of the program should be studied in depth to get a full appreciation for the power of OS-65U.

The format of the files the mailing label program supports are as follows. The first ten characters of each mailing label file contain a pointer to the end of the file so that new entries can be quickly added to the file. Then after the first ten characters, each mailing label entry is allocated a total field length of 120 characters which are allocated as follows. The first 25 characters - the person's name, second 25 characters - the company name, third 25 characters - street, fourth 25 characters - city and state, 10 characters max for zip and 10 characters max for a coded keyword field. Thus, the third mailing label data field starts at 2 times 120 characters plus 10 characters into the file. This file structure might seem somewhat wasteful of file storage space but it is capable of extremely powerful editng, sorting and selective searching entries some of which are exploited in the program, some of which are left to user expansions. The NEW command simply sets up the end of file pointer. The EXIT command simply exits the program closing the file on the way out to insure that the most recent operations are stored on the file. The ADD command falls through to line 500 and utilizes a string of subroutine calls to acquire the string information for the label with full editing capabilities so the user can correct typos as they are made, then stores away each label field as a 120 character indexed field. It constantly updates the end of memory pointer and when the completion of the additions are made, a new end of file pointer is stored away at location 0. The LIST command simply lists out the entries from the beginning of the file to the end of file pointer. By simply changing the subroutine at 1200 or adding several conditional PRINT subroutines, the user can adapt this for any specific line printer. One of the most powerful features of this program is the EDIT command. The EDIT command falls through to line 300 and uses the powerful FIND command. By use of the FIND command in BASIC, any string can be keyed on to bring in the entire mailing label or the entire field, so that if only the zip code or the address or the person's name is known, one can bring in the rest of the information. If one is not sure of the spelling, he can substitute ampersands for part of the spelling or try keying in only part of the word. The use of the FIND command to bring an entire field of several entries has extremely powerful applications in all business applications and opens up the use of Onio Scientific's microcomputer systems in applications in big business data base systems.

Thus, with the EDIT command if a little bit is known about the entry, the entire mailing label can be brought in and selectively edited as desired and then placed back in the file. It should be obvious that this program has a far more general purpose than just mailing labels. By simply changing the field specifications and allocating space and lines as desired, this program could be expanded to a complete data based management system. The optional use of keys in each field can be used to selectively PRINT or LIST labels as desired. For instance, a code character could be put in to specify Northeastern United States or dollar volume of the dealer. A conditional PRINT could be easily programmed which checks each mailing label's keyword and prints it out selectively. Also sorting by any parameter is very straightforward by use of the FIND command and the string equalities in BASIC.

BUS1, BUS2, BUS3, PHONE and LABEL should give a good fundamental understanding of programming file oriented applications with OS-65U. It's believed that because of the power of the INDEX and FIND command and the general ultra high speed operation of the entire system that it should be possible to out perform any other microcomputer system now on the market with properly coded software for any small business application. However, the organization of the file system is of paramount importance in obtaining performance in any file intensive application.

The other programs appearing in the directory are utility programs and are described in the Utility Program section of this manual.

OS-65U FILE DIRECTORY

NAME	TYPE	ACCESS	ADDRESS	LENGTH
DIREC*	OTHER	NONE	25088	1024
BEXEC*	BASIC	WRITE	26112	2048
CREATE	BASIC	WRITE	28160	8192
DIR	BASIC	WRITE	36352	8192
DELETE	BASIC	WRITE	44544	2560
PACKER	BASIC	WRITE	47104	8192
COPIER	BASIC	WRITE	55296	8192
RENAME	BASIC	WRITE	63488	3584
FPRINT	BASIC	WRITE	67072	2304
COPYFI	BASIC	WRITE	69376	7168
FDUMP	BASIC	WRITE	76544	7168
CHANGE	BASIC	NONE	83712	3840
LOAD32	BASIC	NONE	87552	8192
LOAD48	BASIC	NONE	95744	24576
SYSDIR	BASIC	WRITE	120320	4096
MULTI	BASIC	WRITE	124416	2048
BUS1	BASIC	R/W	126464	2048
BUS2	BASIC	R/W	128512	2048
BUS3	BASIC	R/W	130560	2048
ACC1	DATA	R/W	132608	4096
ACC2	DATA	R/W	136704	4096
PHONE	BASIC	R/W	140800	4096
PHODIR	DATA	R/W	144896	8192
LABLE	BASIC	R/W	153088	8192
TLABLE	DATA	R/W	161280	8192

106496 BYTES FREE
25 FILES DEFINED OF 63 POSSIBLE

```
10 REM *** BASIC EXECUTIVE ***
20 REM
30 REM SETUP CONSOLE INPUT DEVICE
40 I% = PEEK (11664)
41 REM
42 REM  LOCKUP SYSTEM
43 FLAG 21: REM  INPUT ESCAPE
44 POKE 2073,96: POKE 14639,0: REM  CTRL C,0
50 POKE 11668, 2^(I%-1)
60 REM
70 REM SETUP CONSOLE OUTPUT DEVICE
80 J% = PEEK (11665)
90 POKE 11686, 2^(J%-1)
100 REM
110 PRINT: PRINT
120 PRINT "OS-65U V1.1"
130 REM
140 REM SETUP FOR MASTER 3300 SYSTEM
150 POKE 61438,0: POKE 61439,0
160 REM
170 REM  GET USER FUNCTION
180 REM
190 PRINT: INPUT "FUNCTION "; FU$
200 IF FU$ = "DIR" THEN RUN "DIR"
202 IF FU$ = "PDIR" THEN RUN "DIR",42
210 IF FU$ <> "UNLOCK" GOTO 280
220 REM
230 REM  UNLOCK SYSTEM
240 REM
250 FLAG 22: REM  INPUT ESCAPE
270 POKE 14639,255: POKE 2073,76: REM  CTRL C,0
280 REM
290 PRINT: PRINT PEEK(132)+PEEK(133)*256-24576; "BYTES FREE": PRINT
300 END
```

```
10 OPEN"ACC1","ANAN",1
15 PRINT
20 INPUT "ACCOUNT NO.";X
30 IF X<0 OR X>199 THEN GOTO 20
35 INDEX<1>=X*20
40 INPUT "CHECK(C) OR MODIFY(M)";A$
50 IF A$="M" GOTO 100
70 INPUT%1,A
80 INPUT%1,B
90 PRINT "ACCOUNT";A,"AMOUNT";$R,B
95 INPUT "ANOTHER(A)";A$
98 IF A$="A"GOTO 15:GOTO 200
99 GOTO 200
100 INPUT "NEW AMOUNT";Y
110 PRINT%1,X
120 PRINT%1,Y
130 GOTO 95
200 CLOSE
```

```
10 PRINT "ACCOUNT DESCRIPTIONS"
20 INPUT "ADD(A) LIST(L) OR NEW(N)";A$
30 OPEN"ACC2",1
35 IF A$="N" GOTO 500
40 INPUT%1,Z
45 INDEX<1>=10
50 IFA$="A"GOTO 300
60 INPUT%1,A
70 INPUT%1,D$
80 PRINT A;TAB(10);D$
90 IF INDEX(1)<Z GOTO60
100 CLOSE
110 END
300 REM
302 INDEX<1>=Z
310 INPUT "ACCOUNT ";N
320 INPUT "DESCRIPTION";D$
330 PRINT%1,N
340 PRINT%1,D$
350 Z=INDEX(1)
380 INPUT "ANOTHER(A)";X$
390 IF X$="A" GOTO 310
400 INDEX<1>=0
410 PRINT%1,Z
420 CLOSE
430 END
500 INPUT"PASSWORD";A$
502 IF A$<>"PASS"GOTO 420
505 Z=10
510 PRINT%1,10
520 GOTO 300
```

```
10 OPEN"ACC1",1
20 OPEN"ACC2",2
30 INPUT%2,Z
40 INDEX<2>=10
50 INPUT%2,A
60 INPUT%2,A$
70 INDEX<1>=20*A
80 INPUT%1,B
90 IF A<>B THEN PRINT"ACCOUNT ERROR":CLOSE:END
100 INPUT%1,C
110 PRINT A;TAB(10);A$;TAB(50);$R,C
120 IF INDEX(2)<Z GOTO 50
130 CLOSE
```

```
10 PRINT"PHONE DIRECTORY"
20 OPEN "PHODIR",1
30 PRINT
40 INPUT "NEW(N) FIND(F) ADD(A) EXIT (E)"; A$
45 INDEX<1>=0
50 IF A$="N" GOTO 200
60 IF A$="A" GOTO 400
65 IF A$="E" GOTO 1000
70 INPUT "NAME"; B$
80 FIND B$,1
90 IF INDEX (1)>1E8 THEN PRINT "ENTRY NOT FOUND" :GOTO 30
100 INPUT %1,C$
110 INPUT %1,C$
120 PRINT "THE NUMBER IS";C$
130 GOTO 30
NEW--200 PRINT"NEW DIRECTORY":PRINT
210 INPUT "NAME";N$
220 INPUT "NUMBER";M$
230 PRINT%1,N$
240 PRINT%1,M$
250 INPUT "DONE(D)";X$
260 IF X$<>"D" GOTO 210
270 PRINT%1,"END"
280 GOTO 30
ADD -400 FIND "END",1
410 IF INDEX (1)>1E8 THENPRINT "EOF ERROR": STOP
420 GOTO 210
EXIT -1000 CLOSE : END
```

```
10 PRINT "MAILING LABLE DEMO PROGRAM"
15 PRINT: INPUT "LABLE FILE"; A$
17 OPEN A$,1
20 PRINT
30 PRINT "FUNCTIONS"
35 INDEX<1>=0
40 PRINT "LIST(L)"
50 PRINT"ADD(A)"
60 PRINT "EDIT(E)"
70 PRINT "NEW(N)"
80 PRINT "EXIT(X)"
90 INPUT A$
100 IF A$ = "X" THEN CLOSE : END
110 IF A$ = "N" GOTO 200
120 IF A$ = "E" GOTO 300
130 IF A$ = "A" GOTO 500
140 IF A$ = "L" GOTO 700
150 GOTO 20
200 INPUT"PASSWORD"; A$
210 IF A$<>"PASS"GOTO 20
220 PRINT%1,10
230 GOTO 20
300 PRINT "EDIT ROUTINE"
305 INDEX<1>=0
310 INPUT "KEY WORD"; K$
320 FIND K$,1
330 IF INDEX(1)>1E8 THEN PRINT "KEYWORD NOT FOUND" : GOTO 300
340 GOSUB 1000
350 GOSUB 1200
360 INPUT "CHANGE(Y OR N)"; A$
370 IF A$ = "N" GOTO 20 '
380 GOSUB 1320
390 GOSUB 1380
400 GOSUB 1440
410 GOSUB 1505
420 GOSUB 1570
425 GOSUB 1630
430 GOSUB 1700
440 GOTO 20
500 INPUT % 1,E
510 PRINT "ADD LABLE ENTRYS AS DESIRED"
520 GOSUB 1300
530 GOSUB 1360
540 GOSUB 1420
550 GOSUB 1480
560 GOSUB 1550
565 GOSUB 1610
570 GOSUB 1700
580 E=E+120
590 INDEX<1>=0
600 PRINT %1,E
610 INPUT "ANOTHER (Y OR N)"; A$
620 IF A$="N" GOTO 20
630 INDEX<1>=E
640 GOTO 520
700 INPUT %1,N
710 E=10
720 GOSUB 1050
730 GOSUB 1200
```

```
[illegible] 20
[illegible] "; A$
[illegible] 50
[illegible] "; A$
[illegible] "; S$
[illegible] 20
[illegible] "; A$
[illegible] M$
[illegible] 20
[illegible] )"; A$
[illegible] 2
[illegible] "; A$
```

```
1600 RETURN
1610 INPUT "CODE FIELD";K$
1620 IF LEN(K$)>9 GOTO 1610
1630 PRINT : PRINT K$
1640 INPUT "ALRIGHT(Y OR N)";A$
1650 IF A$="N" GOTO 1610
1660 RETURN
1700 INDEX<1>=E
1710 PRINT %1,N$
1720 INDEX<1>=E+25
1730 PRINT %1,C$
1740 INDEX<1>=E+50
1750 PRINT %1,S$
1760 INDEX<1>=E+75
1770 PRINT %1,M$
1780 INDEX<1>=E+100
1790 PRINT %1,Z$
1800 INDEX<1>=E+110
1810 PRINT %1,K$
1820 RETURN
```

Getting Started With OS-65U

Place the accompanying disk in "A" drive of Challenger I, II or III with at least 32K of RAM and preferably with a serial terminal. (OS-65U does support the 540 video, however.)

Reset the computer and type a D. After several clicks, the message:

OS-65U Version Number

FUNCTION?

should come out.

Answer DIR (carriage return)

and a directory should be printed.

To RUN any BASIC program in the directory, simply type

RUN"PROGRAM NAME"

To place BASIC programs on disk, first CREATE a file for the program with the CREATE program, then type NEW and program in BASIC conventionally. Once the program is debugged, type SAVE"PROGRAM NAME","PASSWORD" (password optional) to store it on disk.

Once the program is on disk, it can be loaded without being run by the LOAD"PROGRAM NAME"command and can be loaded and executed by the RUN"PROGRAM NAME" command.

To use a data file, first create it with the CREATE program. Then, in any BASIC program use it by assigning that file a Channel Number (1 to 8) with the OPEN"FILE NAME","PASSWORD",CN commands. The file can then be operated on by the INPUT%CN,... PRINT%CN,... FIND"STRING",CN, INDEX《CN》, INDEX (CN) and CLOSE CN statements in BASIC programs.

User Notes on OS-65U

Every extended disk BASIC is in many respects a unique language with respect to file I/O. The following discussion is aimed at the experienced computer user who has dealt with several disk extended BASICS. It is basically a warning list of the little "gotchas" that can cause problems for programmers until they become familiar with this specific disk extended BASIC. The first and foremost warning is to pay attention to error messages. Each error message has something important to tell you when an error occurs. Try to resist the temptation to just go on when an error message is printed out. Take the time to look up the error message and figure out what has been done wrong. Beware of exchanging diskettes mid-stream. OS-65U does not have a "MOUNT" command. The only problem that can be encountered has to do with the fact that directory information for a disk is resident in RAM memory whenever a data file is open. It is possible for the directory information from one diskette to be in RAM after a different diskette has been placed in the drive. This can cause several problems. A good rule of thumb is whenever changing diskettes, reboot the system. This will assure you that the proper directory is resident in RAM at all times.

The system is fairly crash proof with regard to diskette file errors except for the case of attempting to utilize a diskette before initializing it or attempting to use a portion of a C-D74 disk before it has been initialized. These conditions can cause fatal crash to occur which will require a reboot of the system.

Two subtle points in the file system to keep in mind both have to do with using files both in sequential and random access modes. One very important point to remember is that the FIND command works from INDEX specified for a channel or file to the end of that file so that if you want a FIND command to operate over an entire file, be sure to set the INDEX to zero for that channel before using the FIND command. Likewise, it is very easy to find all occurrences of a string in a file by first setting the INDEX to zero and applying the FIND command in a loop which conditionally tests for end of file. With this approach, all occurrences of a particular string can be found. When using files in both sequential and random access modes be careful of the accumulation of inter-record or inter-entry garbage. An example of this problem can be demonstrated in the mailing label program, LABLE, which does not properly handle the situation. The LABLE program utilizes fields or entries of 25 characters maximum for name, company name, etc.

The FIND command operates on this file in a pure sequential fashion so that it "sees" sections of the file which are not normally accessed if the entries are shorter than 25 characters. For example, consider a mailing label entry name, John Doe. With the current program, if this entry were edited to a smaller name such as John, the last name Doe would be left over or left imbedded in the file between the two entries. The normal LIST or PRINT commands would not print out Doe but the FIND command could potentially find the string "Doe" when searching for Doe in some other context. The solution to the problem is to make sure that the inner-entry or inner-record gaps in files are always filled with spaces or nulls. All files are filled with nulls upon initialization and after repacking so the user must simply always replace strings in a file with strings of the same length. This can be accomplished by adding spaces to the right hand side of the string. Do not attempt to accomplish this by adding spaces to the left hand side of the string because these will be stripped off in BASIC as lead in spaces.

User Programmed Disk I/O

The BASIC programmer can perform disk I/O operations by following the steps shown here.

These statements must be executed once to set up for the transfers:

```
POKE 8778,192 : POKE 8779,36     Points USR function to interface
                                 subroutine

POKE 9432,243 : POKE 9433,40     Sets up ISR PUT in interface
                                 subroutine

POKE 9435,232 : POKE 9436,40     Sets up ISR GET in interface
                                 subroutine
```

For each transfer to be performed, the following statements must be executed:

```
CB=9889                                  Defines address of I/O
                                         control block

DH=INT(DA/16777216) : RM=DA-DH*16777216  Place disk address into CB

DM=INT(RM/65536)    : RM=RM-DM*65536

DL=INT(RM/256)      : RM=RM-DL*256

POKE CB+1,RM : POKE CB+2,DL: POKE CB+3,DM : POKE CB+4,DH

Q=256

POKE CB+5,NB-INT(NB/Q)*Q : POKE CB+6,INT(NB/Q)   Place number of
                                                 bytes into CB

POKE CB+7,RA-INT(RA/Q)*Q : POKE CB+8,INT(RA/Q)   Place ram address
                                                 into CB

DEV"A"                                           Select disk device

RW=Ø                                             Specify read;
                                                 RW=1 for write

ER=USR(RW)                                       Perform I/O transfer
```

The user must have previously assigned values for the disk address, number of bytes to be transferred and ram address to the variables DA,NB and RA, respectively.

The maximum range of values for these variables is:

	Floppy Disk	CD-74 Disk
DA	0-275967	0-72898559
NB	0-65535	0-65535
RA	0-65535	0-65535

Of course, an adequate amount of ram must be allocated by the user prior to performing a transfer.

If any error occurs, a non-zero error code will be returned in ER. The following statements might be used for error reporting:

```
IF ER  Ø GOTO 1000
1000 PRINT "*** DEVICE A ERROR"; ER; "AT ADDRESS: DA
1010 POKE 8778,208 : POKE 8779,16   Set USR to function
                                    call error
1020 END
```

It is not necessary to retry disk transfers in an attempt to recover from soft errors; the disk drivers perform an adequate number of retries before returning an error code.

Appendix II

BASIC - DOS Interface Subroutine

The machine language subroutine which is used to interface between BASIC and the 65U disk drivers is an example of how the user programmer should code similar interfaces. An explanation of the subroutine follows.

Sixteen bits of data are passed from a BASIC program as the argument of the USR(X) call. To acquire this data, the machine language code must JSR indirect through location 6 as is done in the first line of the subroutine. This places the 16 bit data value into page zero locations FACLO (hex B2) and FACLO+1. Since the DOS will need to use most of page zero and BASIC's page zero and stack must be saved, the subroutine next calls the swapper. This interchanges all of page Ø and 1 and the registers X,Y and the stack pointer with those saved in the swapper buffer at hex 4700. The registers and page zero and page one currently in use are referred to as the "context". There are two contexts in the system, that of BASIC and that of the DOS. User programmers should generally use the DOS context which provides available page zero locations Ø-F, 50-9F, and CØ-FF.

The subroutine next places the last device number specified in a DEV statement into the transfer control block and pushes a common return address onto the stack for the PUT or GET call to follow.

The parameter, X, passed to this subroutine must be zero for a disk read or one for a disk write. The subroutine tests

the parameter and branches to a JMP to the appropriate routine. The calling sequence for these routines is a JSR followed by a .WORD specifying the address of the transfer control block. (The disk address, number of bytes and ram address must have previously been POKEd by the calling BASIC program.)

After a return from the PUT or GET routine, another JSR SWAP is executed to restore BASIC's context. Then any error code returned by the disk drivers is returned to the calling BASIC program by loading it into the Y (low) and A (high) registers. The indirect JMP through location 8 returns control to the BASIC interpreter.

A0

```
 10 0000          ;     BASIC - DOS DRIVER  INTERFACE SUBROUTINE
 20 0000          ;
 30 0000          FACLO  = $B2
 40 0000          DISCN  = $2668
 50 0000          DUN    = $26A1
 60 0000          GET    = $28E8
 70 0000          PUT    = $28F3
 80 0000          SWAP   = $4907
 90 0000          SWAPB  = $4700
100 0000          ;
110 24C0                 * = $24C0
120 24C0          ;
130 24C0 20E824 BASDOS JSR GETVAR    GET PARAM INTO FACLO
140 24C3 200749       JSR SWAP      SWITCH TO DOS CONTEXT
150 24C6 AD6826       LDA DISCN     SETUP
160 24C9 8DA126       STA DUN        DEVICE
170 24CC          ;
180 24CC A924         LDA #USRXR/256   SETUP
190 24CE 48           PHA               COMMON
200 24CF A9DC         LDA #USRXR-ZZ      RETURN
210 24D1 48           PHA                  ADDRESS
220 24D2          ;
230 24D2 ADB247       LDA FACLO+SWAPB  GET FACLO FROM BASIC'S CONTEXT
240 24D5 F003         BEQ UREAD        ZERO MEANS READ DISK
250 24D7          ;
260 24D7 4CF328       JMP PUT          WRITE DISK
270 24DA 4CE828 UREAD JMP GET          READ
280 24DD          USRXR  =  *-1        RETURN ADR FOR STACK
290 24DD          ZZ      = USRXR/256*256
300 24DD A126         WORD DUN
310 24DF          ;
320 24DF 200749       JSR SWAP         BACK TO BASIC'S CONTEXT
330 24E2 A8           TAY              ERROR CODE TO Y (LOW)
340 24E3 A900         LDA #0           ERROR CODE HIGH IS ZERO
350 24E5 6C0800       JMP (8)          RETURN TO BASIC INTERPRETER
360 24E8          ;
370 24E8 6C0600 GETVAR JMP (6)         TO GET PARAM FROM BASIC
380 24EB          ;
```

LOGICAL CHANNEL #'S USE SCRATCH SPACE STARTING AT 9906 ($26B2) (FOR CH 5)

Appendix III

OS-65U Passwords

All passwords used in the release version of OS-65U are defined on this page.

Utility Program Passwords

All utility programs with limited access (other than R/W) are assigned the password PASS.

The PACKER requires a password prior to initiating the disk packing operation. This password is PACK.

The COPIER requires a password prior to initializing the CD-74 disk. The password is 3300.

Systems Program Passwords

The SYSDIR program password for selection of system 1, the Master system, is SECRET. The sample user systems USER1, 2, and 3 have passwords PW.

Changing Size and Location of the Disk Directory

OS-65U system disks are normally allocated as follows:

Sector	Disk Address	
Ø	Ø	System boot (initialization) code
1	3584	System
2	7168	"
3	10752	"
4	14336	"
5	17920	"
6	21504	"
7 up	25088	Files

The first file within the "Files" area must be the directory for the rest of the files on the disk. The "System" area contains a directly executable copy of all the RAM resident code in the 65U system. Thus, each 65U disk normally holds a complete, fully operational system in addition to utility program and user files.

Since the directory file must be the first file in the file space, it's size - and the number of files that it can hold - is limited once the directory file has been created. Release versions of 65U contain a directory 1024 bytes in length which can hold 1024/16-1 = 63 additional file entries. (16 bytes per entry). If this number is insufficient, it is possible to build a system with a larger directory by following these steps:

Changing the Size of the Disk Directory:

1.) With the COPIER utility program, fully initialize a new disk and copy the System portion of 65U to the new disk.

2.) Using the CREATE utility program, create a directory file on the new disk with the following characteristics:

    Filename: "DIREC*"

    Length: Specify a value equal to 16* (number of files) +16

    File Type: "Other"

    Access Rights: "None"

    Password: As applicable

3.) Proceed to create files on the new disk to accommodate BEXEC* and those utility programs and user files to be included on the disk.

4.) Use the COPYFI utility program to transfer existing programs/ data files to the new disk.

It is also possible to create a "data disk" which has only files storage. To do so, follow these steps:

Changing the Location of the Disk Directory:

275967

PAGE 30

1.) With the COPIER utility program, fully initialize a new disk ~~and copy the System portion of 65U to the new disk.~~

2.) Run the CREATE utility program from an existing 65U system, but type only a carriage return in response to its first question. This leaves the CREATE program in RAM and permits the following change:

3.) Enter the disk page address (disk address/256) of the new directory as three bytes (low, mid, high) into locations

9899, 9900 and 9901, respectively. For example, the lowest address that can be used for a disk directory is 3584. This is disk page address 14,0,0 (low, mid, high) and would be entered this way:

POKE 9899,14 : POKE 9900,0 : POKE 9901,0

Note that since the usual disk page address of the directory is 98,0,0 (25088/256), the second and third bytes are already zero so the corresponding POKEs can be omitted.

4.) Run the CREATE program in RAM by typing only:

RUN

5.) Proceed to create a directory file on the new disk with the following characteristics:

Filename: "DIREC*"

Length: Specify a value equal to 16* (number of files) +16

File Type: "Other"

Access Rights: "None"

Password: As Applicable

6.) Proceed to create files on the new disk as needed. These may be either data files or program files; however, any program files must not make reference to the Systems portion of the disk since it does not exist.

Note that use of any LOAD, SAVE or OPEN commands (whether in the direct mode or as part of a program) must be preceded by specifying the appropriate directory page address. For example, to return toadisk with the directory at the usual address, enter:

POKE 9899,98

as well as the DEV unit select command.

3584

Level II

OS-65U, Level II provides, in addition to the Level I functions, an interrupt driven real time clock and up to 16 terminals with interrupt driven character input. The real time clock is set by the operator at system initialization and can be directly accessed from BASIC. It provides year, month, day, hours, minutes and seconds. A countdown timer is also available. It can optionally be used to time-out an event and cause a real time monitor program (RTMON) to be executed. Each of the up to 16 terminals has its own character input buffer into which data from the associated terminal is placed whenever an input character interrupt occurs. Input of a carriage return causes the input line to be passed to the BASIC program waiting in an INPUT statement. Meanwhile, characters being input from the other terminals are input to their own buffer when typed, so each terminal user appears to have almost exclusive use of the machine.

This system is designed specifically for multi-terminal transaction processing and general inquiry under a BASIC program. A demonstration program, MLABLE, is provided as a sample inquiry processing system.

The LevelII system operates very efficiently on CD-74 based systems and can be used with some speed degradation on floppy only systems. Required hardware is the real time clock option on the 470 floppy disk controller and a 550 or CA-10X board with the desired number of terminal interfaces.

Level III

OS-65U, Level III is a true multi-programming, multi-tasking real time operating system. It features a real time executive program residing in a special 4K memory block at hex D000 and supports up to 16 totally independent tasks running in separate memory partitions. Any of these 16 tasks can be a Level II or Level I system or can be machine code, assembler or other non-operating system related functions. Required hardware is a real time clock, 4K of RAM at hex D000 and multiple memory partitions as desired.

OS-65U

DIREC*	D	NONE	25088	1024
BEXEC*	B	WR	26112	3584
CREATE	B	WR	29696	8192
DIR	B	WR	37888	3584
DELETE	B	WR	41472	3328
PACKER	B	WR	44800	8192