OS - 65U

REFERENCE MANUAL

OHIO SCIENTIFIC
a M/A-COM Company

Ohio Scientific OS-65U Reference Manual

<u>TABLE OF CONTENTS</u>

Introduction

Introduction To The OS-65U Reference Manual

First, we would like to begin by telling the reader what this manual is and what it is not. The purpose of this manual is not to provide a tutorial text or a workbook for OS-65U, nor is it to provide an introduction to computer programming. Rather, this text is a reference manual, intended to be informative to both the new user and the experienced user.

In this manual you will find a summary of enhancements made in this release of OS-65U, overview descriptions of the utility programs and their menu-driven environment, descriptions of the transient utilities available for user-selected system options and extensions, detailed discussions of the line editing and CRT cursor control features of OS-65U, and explanations of commands available for user-modified operating-state configurations.

Much effort has been expended in the system development, user-friendliness, and documentation standards of OS-65U V1.3. We sincerely hope the user community of our Challenger computer systems becomes as excited as we are about this system, and will gain as much by the implementation of OS-65U V1.3 as we have.

At this point, we would like to explain the diskette labeling convention used on OS-65U V1.3 floppy diskettes. The top line of your floppy diskette label reads "OS-65U V1.3" followed by one of the following:

CD-74 / CD-36	for 74 or 36 megabyte hard disc systems or floppy only systems
CD-23	for 23 megabyte hard disc systems
CD-7	for 7 megabyte hard disc systems

The next line of the diskette label will be one of the following:

Floppy Diskette Based	This means that the OS-65U system resident on this diskette is to remain on the diskette, and your computer should be booted from floppy by entering a "D" for the boot command.
Hard Disc Based	This means that the system resident on the floppy disc should be loaded on to the systems

portion of the
proper model hard
disc unit
(CD-74, CD-36, etc.),
and your computer
should be booted
from hard disc by
entering a "H" for
the boot command.

If the second line of the diskette contains the phrase "Files Only", then the diskette does not contain an operating system. Therefore one starts up one's system as normal, and access only the file portion of the "Files Only" diskette. (see COPYFI)

The date on the diskette label is the physical duplication date of the given floppy diskette, not the release date of the operating system. The release date and version number for the operating system will be displayed on the system console after system boot-up.

```
------------------
: Special notice :
------------------
```

CD-74 / CD-36 Floppy Diskette Based and Hard Disc Based

The program called "DSKSET" allows one to configure a floppy based diskette to work with the CD-74 hard disc, the CD-36 hard disc or as a floppy only diskette. DSKSET also permits one to configure a hard disc based floppy to work with the CD-74 hard disc or the CD-36 hard disc.

In order to configure a diskette one must RUN DSKSET. DSKSET will then provide a warning message and ask if you wish to continue. To continue type "Y" <cr>, to abort type "ABORT" <cr>. DSKSET will then ask for a password. Answer this question with "DISC" <cr> (or "ABORT" <cr> to abort). DSKSET will then ask for the DEVice containing the diskette to be modified (answering "ABORT" <cr> will abort). Note that ONLY diskettes may be modified by DSKSET. DSKSET will then insure that the diskette to be modified is a CD-74, CD-36 or floppy only diskette. The next question posed by DSKSET is as follows :

```
1) Set Diskette to CD-74 Type
2) Set Diskette to CD-36 Type
3) Set Diskette to Floppy Only Type ?
```

The third choice will only be presented for floppy based diskettes. One may now enter the number of the desired configuration. DSKSET will then proceed to configure the specified diskette. Upon completion, DSKSET will respond with :

```
Enter a <cr> to continue ?
```

Entering a <cr> will cr one to the system's utilities menu. Entering "STOP" <cr> will force a exit to BASIC's immediate mode.

```
-----------
: WARNING :
-----------
```

It is imperative that one make a copy of the diskettes provided before attempting to use DSKSET. DSKSET modifies the operating system portion on the diskette specified. If a read or write error occurs, the state of the system portion on the diskette is unknown and must be viewed as defective.

```
 -----------
 : WARNING :
 -----------
```

If one instructs DSKSET to configure the diskette as a floppy only diskette, the diskette will not "boot" on a system containing a "powered up" hard disc. Further more, if one attempts to access a hard disc via the DEV command, the system will report a FC error.

```
 -----------
 : WARNING :
 -----------
```

The changes made by DSKSET are not effective until the diskette is "rebooted" or (in the case of hard disc based diskettes) is copied to the system's portion of the hard disc and the hard disc is "rebooted".

Overview of OS-65U

Overview of the OS-65U Operating System

You have made an excellent computer hardware and software choice. Your Challenger computer system, equipped with OS-65U V1.3, is a system which is easy and cost-effective to use in software development, maintenance, modification, and execution.

The programming language of OS-65U is a high speed interpretive BASIC language using the 65Ø2 microprocessor. The BASIC syntax is easy to understand, and the interpretive programming approach makes program writing and de-bugging a simpler and faster task than compiler-oriented languages. Disc input/output procedures in OS-65U are powerful, yet easy to use. Moreover, they are of the same simple syntax whether one is using a sequential or complex random access file structure, or whether one is working with files of a few hundred bytes or a megabyte.

In a similar fashion, OS-65U interpretive BASIC has a strong advantage in solving the problem of software maintenance. One simply uses the on-line program editor to perform a change to source code, and then can directly execute this code to test.

The migration path from a single user system to more complex environments is a straight forward task. OS-65U time-sharing systems will support up to eight users, and networks can support multiple intelligent terminal workstations clustered around up to two hard-disc nodes.

Best of all, the degree of software compatibility between a simple single user system and a complex network time-sharing system is extremely high. If today you develop your software for a one-user floppy-disk based Challenger computer system, you can be sure that, with very minimal changes, tomorrow you will be able to use your software on your Challenger hard-disc based network time-sharing system, or on any configuration midway. Your investment will grow with you.

In summary, we feel that OS-65U V1.3 will save you time and money in software development, maintenance, and utilization. Likewise, as your system needs grow, your initial investment in OS-65U V1.3 will be enough to support your expanded needs.

System Enhancements in Version 1.3

OS-65U Enhancements In Version 1.3

One of the major differences between this version and earlier versions of OS-65U is the magnitude of enhancements, new features, and system extensions found in Verision 1.3. The following is not a discussion of bugs found in earlier releases that were fixed in this release. Rather, we are concerned here with the new, not the old, though often times the two concerns overlap. In regard to the cleanup work done on existing bugs, much has been done, and it is documented elsewhere.

Following will be brief overviews of the new features in OS-65U V1.3.

An extended INPUT mode is introduced in this release. Some of its features are BASIC INPUT commands with data types, field length limits, pre-loaded INPUT strings, and escape sequence and extended control character facilities. In this mode, the BASIC PRINT statement has left/right field justification and field width definition. Also, inherent in this mode is a versatile set of CRT cursor addressing and manipulation controls, and they may be configured for virtually any common CRT terminal currently manufactured.

One is now able to RUN a BASIC program from within another BASIC program with the option to save the current values of the variables defined in the BASIC workspace, thus giving the ability to chain programs together, the result being a virtual program of a very large size. One no longer need worry about available memory as a program size or function limit, and one can document a program with REM statements freely without undue regard to memory loss.

All files are created on integer multiple sector boundaries with integer multiple sector lengths (sector length=3584 bytes). This insures time sharing compatability.

Error trapping and recovery has been significantly enhanced in OS-65U V1.3. One may now, by the use of FLAG commands, route all errors (BASIC and disc errors), or just disc errors, to program line 50000 for custom error-processing. In line 50000 processing, the microprocessor stack is now saved. This facilitates error recovery, as one may now safely return to execution within a subroutine or a FOR-NEXT loop.

With line 50000 error processing enabled, both out-of-memory (OM) and full-stack (FS) error behavior has been improved. These conditions will first CLEAR the variable table and rewind the stack pointer, then jump to line 50000 for error reporting and processing.

If a FLAG has been set to direct error conditions to program line 50000, and if the program has no line 50000, then upon an error condition a jump to the immediate mode is performed.

A new FLAG command will disable or enable the line 50000 error trapping of an overflow (OV) error for numbers greater than 4,294,967,295. This facilitates greater user control over mathematical processing or numeric data entry.

A new error condition, the sempahore stack overflow (SS) error, is being handled. Simply, if a time sharing user tries to set greater than sixteen semaphores at the same time, an SS error will result.

OS-65U V1.3 has as an extension a line editor for use in data and program entry/edit work. This editor gives the operator line recall and various cursor manipulation functions.

The FLAG commands have been greatly extended. Our goal is to make an application program as POKE/PEEK-free as possible, thus the introduction of many new FLAGS. Please note that FLAGs 3, 4, 19, and 20 are no longer functional. Also, executing FLAG n, where n is not a defined FLAG number, will result in a no-operation condition.

On all output devices, the auto carriage return/line feed sequence has been suppressed for a PRINT of a number of characters greater than the terminal width. Also, after a PRINT of 255 characters without the occurance of a carriage return, the BASIC POS (print position) command internal counter will wrap to zero.

In earlier releases the PRINT#5! command was used in time sharing systems to free the printer ownership from a given user. This is still true, but the PRINT#5! command has been expanded as follows:

In a single user environment, PRINT#5! will eject the printer to top-of-form if not at top-of-form. This is true if and only if OS-65U paging is enabled.

In a network environment, the PRINT#5! command behaves the same as in a single user environment.

In a time sharing environment, the above is true, but additionally this command will free the user's exclusive ownership of the printer.

*** One should not use the PRINT#5! as a means of enforcing a page eject to top of form because of the implicit printer-freeing function performed by this command in a time sharing environment.

Two new FLAGS are now available to control printer top-of-form. FLAG 100 is a conditional printer top-of-form control. If the page is not at top-of-form, the printer page will be advanced to top-of-form. FLAG 101 is an unconditional printer page eject. Both FLAGS lock the printer in time sharing, and a PRINT#5! command must eventually be used to free the printer.

Also, in a time sharing environment, if one enters into the immediate mode, the printer page is advanced to top-of-form if not already at top-of-form. This is true assuming one has previously directed some output to the printer.

Please note that if the user disables the OS-65U resident printer paging scheme (in favor of using a printer-controlled paging scheme such as "PRINT CHR$(12);"), then if one is running in time sharing, it is the programmer's responsiblity to make sure that the printer is at top-of-form before freeing it with the the PRINT#5! command.

Upon entry to the immediate mode, the operating system will first dump all dirty buffers to disc. This provides a greater insurance against accidental loss of data or non-desired buffer dumps upon physically switching floppy diskettes.

With a FLAG2 enabled, line 50000 error processing will attempt to write any disc buffers pending to disc. If this cannot be done because of a fatal disc error, the buffer contents will be lost, thus preventing system lockup on an unresolvable disk error loop. Similarly, the same holds true for entry to immediate mode.

A system identification byte has been implemeted whereby a user may PEEK it's value to determine if one is running in a single user, intelligent terminal, time sharing, or network environment.

Utility Programs

OS-65U V1.3 Utility Programs

After boot-up of your computer, the program BEXEC* will perform necessary system initialization functions, and then display a menu of available options as follows:

1)	DIRECtory	This choice will produce a report on your console of disc space usage for a given DEVice.
2)	Print DIRECtory	The report produced is the same as above, but the output is routed to the printer.
3)	Systems Utilities	This set of utility programs is made up of the tasks most used by the end user for file system maintenance, such as creating, deleting, or dumping a file.
4)	Transient Utilities	These are system configuration routines, typically for enabling or disabling operating system options, or for CRT terminal configuration.
5)	Unlock System	By choosing this selection, and by responding with the password "UNLOCK" when prompted, the operator will be put in the immediate mode of OS-65U for programming or command entry. Please note that this should only be done when one is confident with one's working knowledge of OS-65U.

Before beginning to discuss each utility program individually, some common features should be noted.

Each utility can be run by selection from a menu, or by running from the immediate mode. The program names for each utility are given in parenthesis in the sections that follow.

At the end of each utility program's execution, the operator will be given the prompt

"Enter a <Return> to continue ? "

If one enters the string "STOP" followed by a carriage return, the program will exit and the user will be placed in the immediate mode. Entering just a carriage return will return one to the appropriate menu.

The phrase "DEVice" appearing in the prompts means disc storage DEVice. Below is a table of DEVice values for all possible system configurations:

	DEVice Values	
System Configuration	Floppy Disks	Hard Disc(s)
Single User	A thru D	E
Intelligent Terminal	A thru D	E, K thru Z
Time Sharing	A thru D	E
Network	A thru D	E, K thru Z

From time to time one might misguide a given program to an error condition, for example, answering "C" to a DEVice prompt when one has no DEVice "C" on one's system. This could result in an appropriate error message and the entrance to the immediate mode. To get going again, type:

DEV "A"	This should be the DEVice that the OS65U master is being run from.
RUN"BEXEC*"	This will display the main menu.

The above technique should only be used by an experienced user.

For a file created with read/write (R/W) access rights, use the "." character in responding to any prompts for file password. This essentially is the default for no password.

In the majority of system utilities, by answering any prompted input with the response "ABORT", one has an opportunity to abort the utility task and return to the utilities menu.

Systems Utilities

1) DIRECtory (DIR)	This program produces a report of disc space usage for a given, disc DEVice. The information reported is file name, file type, access rights, starting address, length, and address sector boundary condition checks. At the end of the report is summary of disc space used, available, and recoverable. This report is directed to your console.
2) Print DIRECtory (DIR)	The report produced by this procedure is identical to that above, except that the output is routed to the system printer.
3) Create File (CREATE)	This utility allows the operator to create a new file on a given disc DEVice. The operator is prompted for disc DEVice and file name, type, and size information, then the disc DIRECtory is updated for the newly created file.
4) Delete File (DELETE)	In this program the operator is prompted as to what file on what DEVice to delete. Please note that this procedure only flags the file space as being deleted (and recoverable), and it cannot be reused till the PACKER procedure is executed.
5) Rename File (RENAME)	In this program, one can rename both the name and password of a given file.
6) Dump File Contents (FDUMP)	Using this program, one can list the unformatted contents of a given file to the console or printer in either ASCII

	or hexadecimal output mode.
7) Copy From File to File (COPYFI)	One can use this program for file backup or duplication. The file that is being copied to must have been previously created. When specifying the destination file information, by typing an "=" for the "To" file name, COPYFI will use the "From" file name and password information for the "To" file, thereby allowing an easy means to copy a file to a file of the same name on a different disc DEVice.
8) Disc Copier (COPIER)	This program copies the operating system portion, files portion, or both from a given disc DEVice to another disc DEVice. If the destination disc is new and unused, it must be initialized first. Please note that if one accidentally chooses Initialization, by typing "ABORT" for DEVice, one can exit the initialization step without doing any harm.

Other Utility Programs

In addition to those utility programs available as menu selections as mentioned above, there are other utility programs supplied with OS-65U V1.3 as follows:

Recovering Deleted File Space (PACKER)

This program searches for DIRECTory entries marked as deleted, and makes this file space usable by "packing" the disc, i.e., closing in the gaps of deleted file space. A report is produced showing the newly packed disc DIRECTory.

Modifying Disc Contents (CHANGE)

Using this program, one can modify the disc contents on a byte-by-byte basis, for example in applying program patches. This should ONLY be used by experienced programmers.

Print the Contents of a File (FPRINT)

One can print the contents of a given sequential or random access file with this program.

Systems Directory for Hard Disc Systems (SYSDIR)

This program is used to define multiple systems on a single hard disc. It is also used to set disc access limits i.e., to 'enter' into a given defined system.

Configure System for a Serial Printer (PR5TO8)

This program maps logical print device 5 to physical print device 8, thus allowing a serial printer to use the inherent print and paging control built in OS-65U V1.3's parallel printer controller.

Maintain Hard Disc Defective Sectors File (DEFLIS)

One can maintain a file of defective sectors on a hard disc, thus allowing the file creation process to skip over bad spots on the disc.

Load Machine Subroutines into an OS-65U BASIC Program (LOAD32, LOAD48)

These two utilities provide a means to load an OS-65D assembler-produced machine language routine into the workspace of OS-65U. LOAD32 is for a 32K machine, LOAD48 for a 48K machine.

Hard Disk Test (OKTEST) (CD-36 and CD-74 only)

This program is used to test the read/write validity of a hard disc. Please note - This is a DESTRUCTIVE TEST, thus the contents of your hard disc are destroyed by running this program.

Disabling the BASIC Extended Input Feature (INPOUT)

This program removes the extended input and editing features installed in a system by INP$, and re-installs the BASIC functions that were overlaid by INP$. In addition, if common variables are enabled, it will first disable that feature before disabling extended input.

CRT Parameter Fetching (GETCRT)

With the assumptions that the user has extended input enabled, this program provides a means for the user to obtain the resident CRT terminal control codes for use in BASIC application programs. Also provided in this program is a sample subroutine for CRT cursor addressing.

Transient Utilities

Transient Utilities

The transient utility programs configure the environment in which your system is running, i.e., the enabling or disabling of operating system options, and the selection of CRT terminal characteristics for a specific terminal.

It is strongly suggested that these programs be run from menu selection only, and not from the immediate mode. The transient menu program (named "/") has internal logic to handle the disabling of a given option that may be needed before enabling another mutually exclusive option, i.e., disabling the editor before enabling the resequencer. When the transient menu is displayed, included is a summary of what options are currently enabled.

The available transient utilities are as follows:

1) Editor (EDITOR)	Enabling the editor gives the operator line editing features and cursor control for data or program entry/edit.
2) Resequencer (RSEQ)	With this command resident, one has the ability to renumber a BASIC program, optionally specifying line number range and incremental values.
3) Extended Input (INP$)	INP$ enables the extensions to the BASIC INPUT and PRINT commands. Also included is the line editor and CRT controller.
4) Common Variables (COMKIL)	With this option enabled, one has the ability to RUN programs saving the variable workspace memory contents, thus giving the ability to "chain" programs together. Additionally, a KILL verb is added to BASIC which removes variables from the variable table, thus allowing array redefinition and variable space reclamation during program execution.

5) Terminal Setup (BEXEC 2)	Using this program, one can, by simple menu selection, configure the system for a specific CRT terminal to be recognized by the line editor and cursor control functions. One may additionally define a given terminal choice as the system default, to eliminate the need for continual terminal setups. With this feature it is possible for each user in a time-sharing environment to be using a different model CRT and all make use of the editor/CRT functions. The parameters for each defined terminal are stored in the file "CRT Ø".
6) Standard System	This option disables any of the above conditions that might be currently enabled, and re-configures the system to standard OS-65U V1.3.

In various places where the transient utilities are discussed in this manual, there is mention made to the mutual exclusiveness of some of the transient utilities. The table below is a specific summary of the exclusive and inclusive conditions:

Program	Exclusive	Inclusive
RSEQ	EDITOR INP$ COMKIL	RSEQ
EDITOR	RSEQ INP$ COMKIL	EDITOR
INP$	RSEQ EDITOR	INP$
COMKIL	RSEQ EDITOR	COMKIL INP$

Hopefully, after reading the above, one has a feel why it is strongly urged that the user disables/enables all transient utilities by menu selection only, where the management of implicitly required enablement/disablement is maintained for the user.

The Line Editor

The OS-65U V1.3 Line Editor

The operating system extension for enhanced program or data entry/edit capabilities is enabled by running the program EDITOR, and is also included in the extended input program INP$.

The following is a discussion of the functions of the EDITOR. INP$, and its additional functions beyond line-editing, is discussed in its own section.

The EDITOR is terminal independent, in that a given user may configure his running environment for a particular CRT terminal by a menu selection. To give a specific example, in a time sharing environment, one user may be using a Hazeltine 1420 terminal, and another using a DEC VT100, and both users may have the EDITOR features at their disposal, configured to their specific terminal.

The following is a summary of the operator-keyed control sequences for editing and cursor manipulation:

!nnnn	Recalls program line number nnnn for editing, for example !1055 or !12. An attempt to recall a non-existent line number returns the next highest numbered line if any.
!	Recalls the next sequentially available line for editing, after defining a starting position with the above technique.
!!	Recall the same line again for re-editing.
@	Commercial "at" sign or shift-P. Delete the line being currently edited. This is a CRT function only, not a program function. In other words, this command erases the current line from the CRT screen, but leaves the line in memory untouched.
Control F	Non-destructive cursor move to the front of the line.
Control R	Non-destructive cursor move to the rear of the line.
Control I	Tab eight character positions to the right. One may not tab beyond the current end position of a given line.
Control T	Toggle between character insert/overstrike mode. In the insert mode, each character

entered from the keyboard is inserted into the line, where as in the overstrike mode, the character entered replaces the character previously appearing in the current cursor position.

The insert/overstrike mode may also be toggled as follows:

```
POKE 23721,Ø   :REM insert mode
POKE 23721,255 :REM overstrike mode
```

Rubout or Delete	This is the delete character code, or destructive backspace.

Non destructive cursor forward and back space codes are also available. As a convention we have chosen to use the control H (ASCII 8) for back space, and the Control L (ASCII 12) for forward space. On the Hazeltine 142Ø these are the left and right arrow keys respectively.

The control codes for the CRT screen manipulation functions are stored in the DMS-PLUS master file "CRT Ø". The codes used by the EDITOR are as follows:

code recognized as incoming forward space
code recognized as incoming back space
code(s) to be echoed to cause a forward space
code(s) to be echoed to cause a back space

Extended Control Characters For The Editor

One may define a table of extended control characters to be recognized as INPUT terminators. After an INPUT command, one can check to see if a given control character was entered, then respond accordingly.

		Values	
	Address	Off	On
Extended Control Enable	23722	Ø	255
Control Character Table (6 positions)	23724	255	ASCII value
	23725	255	" "
	23726	255	" "
	23727	255	" "
	23728	255	" "
	23729	255	" "

Report Table Position 23723	The value returned is either a zero for a normal carriage return INPUT termination, or the control character table position (1-6) if INPUT was terminated by an extended control character.

By POKEing the desired character ASCII values into the above defined table, and by enabling extended INPUT control by a POKE 23722,255, one may interrogate the result of an INPUT as follows:

```
     POKE 23722,255  : REM enable extended control
     INPUT QA$
     EC=PEEK(23723)  : REM get table position if any, else Ø
     ON EC GOTO C1,C2,C3,C4,C5,C6
     PRINT"input terminated by a carriage return"
     END

C1
  :
C2
  :
C6
  :
```

The POKE 23722,255 should be done immediately before each INPUT statement for which extended control is to be recognized. The operating system zero's the contents of location 23722 following an execution of an INPUT command.

Escape Control Character Definition For The Editor

One may make use of the two character escape key-in for customized applications as follows:

```
     POKE 2373Ø,255   : REM enable escape check
     INPUT A$
     ES=PEEK(23731)   : REM get returned escape character

     IF ES=ASC("1") THEN PRINT "Escape 1 was entered"
     IF ES=ASC("2") THEN PRINT "Escape 2 was entered"
```

ES is the ACSII value of the character entered immediately following the escape character. If the escape was not entered, ES will have a value of zero.

The system resets the value of location 2372Ø to a zero after each INPUT. Therefore, one must POKE 2373Ø,255 immediately before each INPUT command where one wishes to check for an escape key-in.

If two escape keys are entered in a row, the net result is a cancellation of the escape sequence.

NOTE - If both the extended control and escape character features are enabled together, the escape key-in check is performed first, followed by the check for any control characters.

The EDITOR may be disabled by running EDITOR from the immediate mode and choosing the disable option, or by choosing an appropriate option from the transient utilities menu.

Extended Input Mode

OS-65U V1.3 Extended Input Mode

The extended input mode is enabled by running the utility program INP$, either from the transient utilities menu or from the immediate mode. INP$ has many entry points, mostly to manage the disablement of any enabled system extensions which might be mutually exclusive with INP$. Because of this, it is advisable to run this program from the transient utilities menu, where the entry point management is automatically handled.

Enabling this system extension gives the user the benefits of extended INPUT and PRINT facilities, and of the line editor and CRT controller which may be configured for any one of a number of CRT terminals currently available.

With INP$ enabled, the syntax for the BASIC INPUT command is as follows:

```
INPUT%n,[A,A$] QA$      n=channel number

INPUT#n,[A,A$] QA$      n=device number

INPUT "prompt string", [A,A$] QA$
```

where

A = maximum length of string to be input.

A$ = data type of the string. Types currently defined are

A - ASCII. This form allows INPUT of any legal ASCII character, with leading spaces maintained, i.e., not thrown away as in normal BASIC INPUT.

I - Integer. This form accepts characters "Ø" through "9, "+", and "-" only. Leading spaces are stripped off the preloaded string value.

C - Cash. This form accepts a real number in the form nnn.nn, that is, a real number with two digits to the right of the decimal point if a decimal point is in fact entered. Leading spaces as above.

F - Floating point. As above, but there may be any number of digits to the right of the decimal point.

QA$ = pre-loaded string to be input. The original contents of this string, if any, will be displayed on the CRT at INPUT time.

Using this type of INPUT command, strings may be INPUT of a length up to 254 characters maximum.

Consider the following example of extended INPUT:

```
CA$="1.23"              :REM preloaded value
PRINT"Enter cash value ";
INPUT [6,"C"]CA$        :REM cash field, 6 characters max.
```

The operator will see "Enter cash value 1.23" on the console, with the cursor positioned on the "1" character. If the operator enters an illegal character, say an upper case "A" for example, the ASCII(7) bell character will be printed to the console, and the "A" will not be accepted or echoed.

Let us assume in this example that we are in insert mode. The operator now presses the "8" key, and "81.23" can be seen on the console. If a return is then entered, then the value of CA$ after INPUT is "81.23".

Let us consider another example using this same case. The operator enters shift-P (@) to delete the current pre-loaded value of CA$. The operator now sees "Enter cash value " with the cursor to the right of the prompt word "value". The operator enters "5.23.4" then a return. This is an illegal INPUT sequence for cash, and the operator will be alerted to this with the console bell sounded, and the cursor positioned on the rightmost decimal point in the string "5.23.4". The operator now deletes this decimal point with the delete or rubout key, and with "5.234" displayed on the screen, now presses return. This is still illegal in that there are now three digits to the right of the decimal point. The bell will be sounded, and the cursor will be over the "4" character. Finally, the operator deletes this character, and presses return, and INPUT is terminated with CA$ having the value of "5.23".

One is encouraged to experiment with the INPUT[ ] command, as the familarity with system behavior can be comprehended much better with experimentation than with written examples as above.

Thus the INPUT command can now be more conditionally controlled by the programmer, with data types and field length specifications.

WARNING - The INPUT[ ] statement does not direct a carriage return/line feed sequence to the console upon execution termination. Rather, the cursor is positioned to the leftmost position of character INPUT, i.e., to the right of any prompting messages.

The syntax for the BASIC PRINT command is as follows:

```
PRINT [A,A$] QA$

PRINT#n,[A,A$] QA$    n=device number

PRINT%n,[A,A$] QA$    n=channel number
```

where

```
A  = print field length
A$ = "L" for left justified
     "R" for right justified
```

Please note that PRINT statement parameters may be any combination of variables or literals, thus the following are all valid PRINT statements with INP$ enabled.

```
PRINT [3,TY$] QA$         PRINT [FL,"R"] QA$
PRINT [3,"L"] QA$         PRINT [3,"R"] "1ØØØ-1Ø1"
PRINT#DV,[FL,"L"] QA$     PRINT%CH,[FL,"R"] QA$
PRINT#5,[FL,"L"] QA$      PRINT%1,[1Ø,"R"] "HELLO"

PRINT #DV,[1Ø,"R"]"1Ø2Ø2";X;TAB(45);[2Ø,"R"]H$
```

The statement

```
PRINT#5,[5,"R"]"XX";[4,"L"]"YY"
```

will direct the following string to the line printer:

```
"   XXYY  "
```

This print feature should be used instead of the more common BASIC statement of the form PRINT RIGHT$("   "+QA$,FL) because the concatenation of strings generates "garbage" in the workspace which eventually the system will have to stop and clean up resulting in a "garbage collection" delay.

The contents of memory location 12Ø98 is the ASCII value of the justified PRINT pad character, normally a space (ASCII 32). One may modify this as follows:

```
POKE 12Ø98,ASC("*")
```

The above example changes the pad character to an asterisk.

Please remember that this pad character feature is ONLY available when INP$ is enabled.

WARNING - If the length of the string to be printed is greater than the specified field width, a long string (LS) error will occur.

INP$, upon execution, configures itself for the CRT terminal currently defined as the system default. (This default may be changed by the transient utilities CRT configuration routine). The paramaters for the CRT terminals are stored in the DMS-PLUS master file named "CRT Ø". The parameters maintained are:

code recognized as incoming forward space command.
code recognized as incoming back space command.
code(s) to be echoed to cause a forward space.
code(s) to be echoed to cause a back space.
code(s) to be echoed to address the cursor.
code(s) to be echoed to clear the screen.
code(s) to be echoed to clear to the end of screen.
code(s) to be echoed to clear to the end of line.
code(s) to be echoed to set foreground.
code(s) to be echoed to set background.

These codes are used by the line editor where applicable, and can be used in application programs for CRT screen manipulation. A sample use of the codes can be found in the program GETCRT.

It should be noted that INP$ inherently enables the line editing extension of OS-65U. A complete discussion of the line editing facilities can be found in the EDITOR detail section of this manual.

INP$ may be disabled by running the program INPOUT, or by choosing to enable a mutually exclusive system option from the transient utilities menu, such as the editor or resequencer.

WARNING - If other than a string variable is used as the object of an INP$ INPUT or PRINT error a type mismatch (TM) error will occur.

WARNING - It should be noted that INP$ is an overlay to standard OS-65U V1.3 BASIC, and therefore modifies some features of BASIC. Specificly, with INP$ enabled, the following BASIC commands are no longer available:

LOG	SIN	TAN
SQR	RND	ATN
EXP	COS	^ (exponential operator)

Attempting to use these functions with INP$ enabled will produce a syntax (SN) error.

Common Variables Mode

OS-65U V1.3 Common Variables Mode

The common variables mode is enabled by running the utility program COMKIL, either from the transient utilities menu or from the immediate mode. As always, unless one is absolutely certain of procedure, run this utility by menu selection only.

Enabling this system extension gives the user the ability to save variable values (that is, the entire variable workspace) upon RUNning a program from another program, thus giving the ability to chain programs together for a virtual effect of much larger programs than can actually fit in the memory workspace. Also, with COMKIL enabled, the BASIC NULL command is replaced with the verb KILL, which allows one to free up memory taken by given variables, thus deleting their definition.

With COMKIL enabled, the syntax of the RUN command is:

```
RUN[PR$,PW$,LN] where

PR$=program name             this form will
PWS=program password         retain the variable
LN =line number              workspace values.

      or

RUN PR$,PW$,LN               these forms are those
RUN PR$,LN                   which do not retain
RUN PR$                      variables.
```

Consider the following program examples:

```
10 REM PROGA                  10 REM PROGB
20 X=100                      20 PRINT X
30 RUN ["PROGB","PASS",10]    30 END
40 END
```

First, the operator RUN's PROGA with the command RUN "PROGA". Line 30 will RUN PROGB, retaining the values of variables in PROGA, in this case, the value of X of 100. In PROGB, the printed value of X will be 100, still retained from PROGA.

The syntax of the KILL command is:

```
    KILL A,B,I, ...P$,...    Kill specific simple variables.
    KILL A(),B(),P$()        Kill specific array variables.
    KILL *                   Kill all simple variables.
    KILL (*)                 Kill all array variables.
```

After KILLing an array variable, it is possible to reDIMension that variable again. For example:

```
DIM A(20,20)
:
:
:
KILL A()          At this point the array A
:                 no longer has any DIMension
:                 or value.
:
DIM A(100,100)
```

WARNING - One should NEVER use the KILL command from within a FOR-NEXT loop, or disastrous results will occur.

When one is in the immediate mode of BASIC, it is easy to detect the presence of COMKIL. If the screen prompt characters are upper case "OK", then COMKIL is not on-line. If the prompt characters are lower case "ok", then COMKIL is on-line.

To disable the common variables extention, one may run COMKIL from line 5000, (i.e., RUN"COMKIL",5000) or from the transient utilities menu by choosing any choice other than common variables.

COMKIL is an overlay to BASIC, and the following functions are not available when COMKIL is enabled:

```
FNA      DEF
```

Attempting to use the above functions with COMKIL enabled will produce a syntax (SN) error.

WARNING - BASIC disc program files must start on sector boundaries in order to be used with COMKIL.

Flag Commands

FLAG Commands

The FLAG commands are used to enable or disable certain system features. They may be used either in the immediate mode or within a BASIC program. The form of the FLAG command is:

```
FLAG n
```

Where n is one of the following:

1 Disables the close-files-on-error and the close-files-on-immediate-mode feature.

2 Enables the close-files-on-error and the close-files-on-immediate-mode feature. With this feature enabled, if one encounters an error condition, or one exits to the immediate mode, any channels currently opened will automatically be closed. Before closure, any buffers pending to be written to disc will be written providing that the given error condition does not prevent this from occurring.

5 Enables user programmable disc EOF action, that is, set the channel INDEX to a value greater than or equal to 1E9 upon encountering a disc file end-of-file condition.

6 Enables program abort and system error message upon disc end-of-file condition.

7 Enables BASIC statement trace. With this feature enabled, each program line executed will be displayed on the console.

8 Disables BASIC statement trace.

9 Enables user programmable disc error action. With this option enabled, program execution will jump to line 50000 upon encountering a disc error condition. This can be exclusively overridden by a FLAG 23, which enables both disc and BASIC errors to be processed at line 50000.

10 Enables program abort and system error message upon encountering a disc error condition. This may be overridden by a FLAG 24, which directs both disc and BASIC errors to the immediate mode.

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. Using this option allows one to LIST a program or parts of a program

to a data file, then later INPUT this data file to another program, thus providing a convenient way for program merge.

14 Disables "INPUT%n," command file operation.

15 Allows the characters comma (","), and colon (":") to be treated as valid in an INPUT or statement. WARNING - READ statements will generally not work with this FLAG enabled, given that most READ statements have delimiting commas within them.

16 Allows the comma, and colon to be treated as INPUT or READ delimiters or terminators.

17 Disable carriage return/line feed upon terminating an INPUT or PRINT. This option is a convenient way of producing an INPUT without carriage return/line feed, however, it should be noted that this disabling is effective on ALL I/O DEVices. The most advisable way to use this option in conjunction with FLAG 18 is as follows:

```
FLAG 17        Turn off CR/LF
INPUT A$
FLAG 18        Turn on CR/LF immediately after
               INPUT
```

```
* * * *WARNING* * * * *
*   ALL DEVICES ARE   *
*   AFFECTED BY THIS  *
*   FLAG, I.E., ALL   *
*   CR/LF SEQUENCES   *
*   SUPPRESSED.       *
* * * * * * * * * * * *
```

18 Enable carriage return/line feed upon terminating an INPUT or PRINT.

21 Disable input escape on carriage return. With FLAG 21 enabled, if an operator enters just a carriage return for an INPUT statement, the system response will be a "REDO FROM START" message followed by a "?" prompt for re-entering the expected INPUT value. This may be overridden by a FLAG 27.

22 Enable input escape on carriage return. With FLAG 22 enabled, a response of just a carriage return to an INPUT statement will cause a jump to the immediate mode. This may be overridden by a FLAG 28.

23 Enable a jump to program line 50000 upon the occurrance

of any or all error conditions (this includes both BASIC and disc errors). FS and OM errors first CLEAR the variables and rewind the stack then jump to line 50000. If FLAG 23 is enabled and the program has no line 50000, the result will be a jump to the immediate mode. This FLAG may be overridden by FLAG 9.

24 Enable a jump to the immediate mode upon the occurrance of any or all error conditions (BASIC and disc errors).

25 Disable control "C" termination of BASIC program execution.

26 Enable control "C" termination of BASIC program execution. With this FLAG on, pressing the console control and "C" characters will cause termination of an executing BASIC program, and a jump to the immediate mode.

27 Enable a null input (carriage return only) as valid INPUT sequence.

28 Disable null input sequence on INPUT. With this FLAG on, a null input will produce a jump to the BASIC immediate mode, unless overridden by a FLAG 21.

29 Disablement of trap-overflow condition.

30 Enablement of trap-overflow condition, specifically, generate an OV error for numbers greater than 4,294,967,295.

100 Perform a conditional top-of-form eject on print device #5. If page is at top-of-form, no eject will be done. Locks device #5 in a time sharing environment. This is available only if the system paging feature is enabled.

101 Unconditional top-of-form page eject. Availability and time sharing behavior as above.

Please note that FLAGs 3, 4, 19, and 20 are no longer functional in this release of OS-65U. Also, executing FLAG n, where n is not a defined FLAG number, will result in a no-operation action.

The following is a list of FLAGs set by BEXEC*.

The conditions set by these FLAGs duplicate the start-up of the operating system.

FLAG 1	disable close-files on error or imm mode
FLAG 6	enable abort & err msg on disc EOF
FLAG 8	disable BASIC statement trace
FLAG 1Ø	enable abort & msg on disc error
FLAG 12	disable space suppression on numeric output to file
FLAG 14	disable 'INPUT%n, " command file operation
FLAG 16	no ':' ',' '&' on INPUT
FLAG 18	enable cr/lf on INPUT or PRINT
FLAG 22	disable input escape on carriage return
FLAG 24	jump to imm. mode on al error conditions
FLAG 25	disable control/C stopping
FLAG 27	enable null on input
FLAG 29	disable trap of overflow error

Summary of BASIC Commands

OS-65U V1.3 BASIC Commands and Reserved Words

Command	Syntax	Description
ABS	ABS(X)	Returns the absolute value of the function argument.
AND	X AND Y	Logical AND operator.
ASC	ASC(X$)	Returns the decimal ASCII value of the first character in the string argument.
ATN	ATN(X)	Trigonometric arctangent function with the argument in radians. Not available when INP$ transient utility is enabled. The range of values for the ATN function argument is -1 to 1.
CHR$	CHR$(I)	Returns a one character string, whose decimal ASCII value is that of the argument.
CLEAR	CLEAR	Clears the variable table and RESTOREs the DATA pointer.
CLOSE	CLOSE CLOSE n	Closes an open disc file by dumping the disc buffer and freeing the channel n (1-8). CLOSE with no channel number specified closes all open channels. A CLOSE n command, where n is a channel that has not previously been opened, will produce an error.
CONT	CONT	Continue execution of a program that has been halted either by a control C key-in or a STOP statement.
COS	COS(X)	Trigonometric cosine function with the argument in radians. Not available when INP$ transient function is enabled.

DATA	DATA 4, 72, "HI"	Provides data elements for READ statements. Strings may appear quoted or unquoted. If unquoted, leading blanks are ignored and trailing blanks are included.
DEF	DEF FNA(X)=X+SIN(X)	Define function statement, where A and X are simple variable names. Not available when COMKIL transient utility is enabled.
DEV	DEV "A" DEV D$	Specifies which disc DEVice is to be currently on-line. The argument must be a single character string literal or variable. The possible values for DEV are: Single User: A-D,E. Intelligent Terminal: A-D,E,K-Z Time Sharing: A-D,E. Network: A-D,E,K-Z.
DIM	DIM A(2Ø), B$(1Ø,5)	Dimension statement for subscripted variables.
END	END	Terminates program execution. This statement need not appear in a program at all, nor necessarily as the last statement in a program.
EXP	EXP(X)	Exponential function of e (2.71828...) raised to the power of the argument. Not available when INP$ transient utility is on-line.
FIND	FIND "LOAN",2 FIND A$, CH	High speed search for the string expression (first argument) in the disc file opened on channel number (second argument). Search starts at current INDEX of that channel. If found, returns INDEX of the found location in the file, else returns INDEX value of

		greater than or equal to 1E9.
FLAG	FLAG 3	Enable the system option defined by the flag argument number.
FN	DEF FNA(X)=2+X Y = FNA(X)	Function name, of the form FN followed by a variable name. Not available when COMKIL transient utility is enabled.
FOR	FOR I = 1 TO 5 FOR J = A TO B STEP C	FOR-NEXT loop range definition verb.
FRE	FRE(X)	Returns the number of bytes of memory workspace available that are unused. X is a dummy variable.
GOSUB	GOSUB 15Ø ON X GOSUB 1Ø,7Ø GOSUB A	Execute a BASIC subroutine beginning at the line number equal to the numeric argument.
GOTO	GOTO 15Ø ON X GOTO 1ØØ, 2ØØ	Unconditional transfer of program execution to the line number equal to the argument.

IF	IF X = 1 THEN GOTO 5Ø
	IF I>1 THEN I=1
		Conditional statement execution verb.

INDEX	INDEX<CH>=Ø N=INDEX(CH)	Set or equate an open disc channel's file index position. INDEX<n> is an index assignment, INDEX(n) is an index equate. In an INDEX<n>=x, the value x must be a non-negative integer.
INPUT	INPUT A INPUT "NAME";N$ INPUT#1, B INPUT%3, F$ INPUT [3,"A"] QA$	Obtain data from console keyboard, or from specified input device or disc file. The INPUT[ ] form of this command is available only with INP$ transient utility enabled.

INT	I = INT(X)
	I = INT(3.1415)
		Returns the greatest integer less than or equal to the numeric argument.

KILL	KILL A A$ KILL C(), B$() KILL * KILL (*)	Eliminate variables from the program variable table. Arguments may be specific simple variable names such as A, specific array variables such as C(), * to KILL all simple variables, or (*) to KILL all array variables. Available only when COMKIL transient utility is enabled.
LEFT$	B$ = LEFT$(A$,5)	Returns the leftmost substring of a given string. First argument is a string expression, second argument is a positive arithmetic expression.
LEN	L = LEN(A$)	Returns the length of the string expression argument.
LET	LET X = Y+1	Assignment statement preface operator. Optional in assignment statments.
LIST	LIST LIST 1-5Ø LIST -5Ø LIST 1ØØ- LIST#DV,1Ø-2Ø LIST%CH	Program listing verb. Examples to the left illustrate syntax for complete listing, listing of line number ranges, and listing to a specific output device or disc channel.
LOAD	LOAD "PROG1" LOAD "P1","PASS"	Command to load a program from disc to memory. Arguments are program name, and password if required.
LOG	LOG(X)	Returns the natural logarithm (log to the base e) of the numeric argument. Not available with INP$ transient utility enabled.

MID$	A$ = MID$(B$,2,4)
	A$ = MID$(B$,7)
		Returns a middle substring of a string argument, with a specific starting character, and either a specified length or implicitly to the end of the string.

NEW	NEW NEW 3584	Reset all program workspace pointers, i.e., start with

		a clean workspace for entry of a new program. The form NEW n, where n is a positive number, reserves an area of n bytes at the beginning of program workspace for custom programming use, such as machine language subroutines or disc transfer buffer space.
NEXT	NEXT NEXT I NEXT I,J	Terminating statement range verb for FOR-NEXT iterative loops. Jumping in and out of FOR-NEXT loop statement ranges should be avoided.
NOT	NOT X NOT (A AND B)	Logical negation operator.
NULL	NULL 8	Inserts Ø to 255 zeros (null characters) at the beginning of each string output by a LIST or PRINT command. Not available when COMKIL OR RSEQ transient utility is enabled.
ON	ON X GOSUB 5Ø, 1ØØ ON E GOTO L1, L2, L3	Conditional transfer statement verb. In the second example, control is transferred to program line L1 if value of E is 1, and line L2 if value of E is 2, and so on.
OPEN	OPEN "FNAME","PASS",3 OPEN "NAE",1 OPEN FN$,PW$,CH	Open a data file on a given channel for program disc access. Arguments are file name, password (if required) and channel number (1-8).
OR	A OR B	Logical OR operator.

PEEK	C = PEEK(23468)
	I = PEEK(J+1)
		Returns the contents of a memory location. Argument is the memory address in decimal.

POKE	POKE 2Ø48,199 POKE J,I+64	Stores a value into a memory location. The first argument is the memory address, and the second location is the value to be stored, between Ø and 255 inclusive.

POS	POS(X)	Returns the print position of the last character printed before the call to POS. X is a dummy variable. Returns a value between Ø and 255 inclusive.
PRINT	PRINT PRINT A PRINT A,C;B$ PRINT#5;PRINTER PORT" PRINT%CH,"DISC OUTPUT" PRINT [1Ø,"R"] A$	Output command for screen, printer, any other output device, or disc file channel. The PRINT[ ] form is only available when INP$ transient utility is enabled.
READ	READ R, A$	Read, from DATA statements, the value of the variables appearing as arguments.
REM	REM HERE IS A COMMENT	Remark or comment initiator. All text after a REM is ignored on a given program line, i.e., it is not executed as BASIC code.
RESTORE	RESTORE	Reset the pointer in a DATA list to the first DATA item.
RETURN	RETURN	Exit verb from a BASIC GOSUB subroutine. Control is transferred to the program location immediately following the GOSUB command that initiated the subroutine execution.
RIGHT$	A$ = RIGHT$(B$,3)	Returns the rightmost substring of of a string argument.
RND	I = RND(X)	Returns a random number between Ø and 1. If the argument is negative, it will be interpreted as a seed value. If it is zero, the function will return the last generated random number again. If the argument is positive, a random number will be returned based on the previously defined seed. Not available when INP$ transient utility is enabled.
RSEQ	RSEQ OL,NL,IN RSEQ OL,NL	Renumbers a BASIC program Syntax reads as follows:

	RSEQ ,NL,IN RSEQ ,,IN RSEQ ,NL RSEQ OL,,IN RSEQ OL RSEQ	starting at old line numbr OL, resequence with new line number NL, in increments of IN. Any permutation of of parameters being present is permitted, noting to keep commas as delimiters where needed. Only available when the RSEQ transient utility is enabled.
RUN	RUN RUN 2ØØ RUN "PGM" RUN "PGM",2ØØ RUN "PGM","PASS" RUN P$,W$,LN RUN[PN$,PW$,LN]	Initiate execution of a BASIC program, either resident in memory or loaded from disc. The RUN[ ] example is the syntax for RUNning a program with saved variable values. It must have program name, password, and line number. The RUN[ ] command is only available when the COMKIL transient utility is on-line.
SAVE	SAVE SAVE "PGM" SAVE "NAME","PASS"	Stores the current program in memory onto a disc file whose name and password are the command arguments. If no arguments are given, then the disc file SAVEd to is the same as the file name used in the last LOAD command executed.
SGN	S = SGN(X+1)	Returns the sign of a numeric argument, i.e., +1 for a positive valued argument, -1 for a negative valued argument, Ø for a zero valued argument.
SIN	S = SGN(AN)	Trigonometric sine function with the argument expressed in units of radians. Not available when INP$ transient utility is on-line.
SPC()	SPC(3)	Used to print spaces inserted in output. PRINT SPC(n) will print n spaces.
SQR	R=SQR(W)	Square root function. Not available when INP$ transient

		utility is on-line.
STEP	FOR I = 1 TO 5 STEP 3 FOR I = 5 TO 2 STEP 1	FOR-NEXT incremental value definition verb. Care should be taken not to explicitly assign a value to the FOR-NEXT increment variable (I is the given examples), as this may destory the expected implicit looping.
STOP	STOP	Interrupt program execution and jump to immediate mode. The program execution may be continued by a CONT command issued from the immediate mode.
STR$	A$ = STR$ (N)	Converts a numeric argument to it's string equivalent. For example, STR$(1.2) = " 1.2", the first character being the sign (blank for positive, - for negative).
TAB	TAB(7)	Tabular spacing function used in printed output. Character positions are relative to zero.
TAN	T = TAN(A)	Trigonometric tangent function, argument in radians. Not available when INP$ transient utility is on-line.

THEN	IF A=B THEN 2ØØ
	IF C<2 THEN C=2
		Conditional statement execution directive verb.

TO	FOR I = 1 TO 3	FOR-NEXT range definiton verb.
USR	Y = USR(X)	Call to user-defined machine language subroutine resident in memory. The argument is a single parameter that can be passed to the user-routine.
VAL	N = VAL(A$)	Returns numeric value of a string argument, or zero if the argument is non-numeric.

WAIT	WAIT I,J WAIT I,J,K	Halts program execution, i.e., causes a program to "wait", until a particular bit or bits in memory is set. In the first example, the WAIT function reads the status of memory location I, then ANDs the result with value J until a non-zero result is obtained. The second example reads the status of memory location I, exclusive ORs that value with K, then ANDs that result with J until a non-zero result is obtained.
WAIT CLEAR	WAIT CLEAR 2	Clears, or unsets, a semaphore in a time sharing environment. No operation in a single user environment.
WAIT FOR	WAIT FOR 21	Sets a semaphore in time sharing, no-operation in single user.

BASIC Commands and Reserved Words - By Function

Program Execution Control

The following commands control BASIC program execution flow such as braching, looping, and subroutine calls.

```
CONT    GOTO    RUN    USR
END     IF      STEP   WAIT
FLAG    NEXT    STOP   WAIT CLEAR
FOR     ON      THEN   WAIT FOR
GOSUB   RETURN  TO
```

Disc Input/Output Commands

Please note that the disc I/O commands respect the access rights and file type assigned to a given file at CREATE time. The possible access rights are:

Read/Write	without password.
Read	without password.
Write	without password.
None	no access without password.

The possible file types are:

BASIC	- BASIC program file.
Data	- General data storage.
Other	- DIREC*, special function system files.

Consider the following example:

```
OPEN "DATFIL",1
PRINT%1,"HELLO"
```

If file DATFIL has read-without-password access rights, then the above code will produce an access error because the file was not opened with the password specified, i.e., OPEN "DAFIL","PASS",1.

```
CLOSE   LOAD
DEV     OPEN
FIND    PRINT
INDEX   SAVE
INPUT
```

General Input/Output Commands

Included are commands for console, printer, memory, and disc I/O.

```
 DATA    PRINT
 INPUT   READ
*NULL    RESTORE
 PEEK    SPC
 POKE    TAB
 POS
```

Logic Functions

```
AND
NOT
OR
```

Mathematical Operations

Please note that some of these functions are not available with the INP$ and COMKIL transient utilities enabled.

```
 ABS    *FN     *SIN
*ATN     INT    *SQR       * avail. only when
*COS    *LOG    *TAN         trans. util.
*DEF    *RND     VAL         disabled.
*EXP     SGN
```

Program Execution and Manipulation

```
FLAG   REM
FRE  **RSEQ          ** avail. only when
LIST   RUN              trans. util.
LOAD   SAVE             enabled.
NEW    USR
```

String Variable Manipulation

```
ASC     MID$
CHR$    RIGHT$
LEFT$   STR$
LEN
```

Variable Definition and Manipulation

```
  CLEAR   PEEK
  DIM     POKE
**KILL    REM
  LET
```

Programmer's Reference Guide

Function: CONSOLE CONTROL CHARACTERS

Keywords: CONTROL C,S,Q,W,D, CONSOLE

The serial console device provides a number of control character commands for controlling output to the console and BASIC execution. These commands are listed below.

Control - C --	Stops a BASIC program listing or execution at the end of the current statement if this option has been enabled with FLAG 26.
Control - S --	Stops all output pending input of a Control 'Q'.
Control - Q --	Restarts output that was stopped with a Control 'S' or Control 'D'.
Control - O --	Causes output to be 'thrown away' pending input of another Control 'O' or entry into the immediate mode.
Control - D --	Limits output to one screen at a time, then stops pending input of a Control 'Q' (Console paging.) This is only looked for during output to the console.
Control - W --	Terminates the paging of output that was initiated by a Control 'D'

Function: INPUT/OUTPUT DISTRIBUTION

Keywords: I/O DISTRIBUTOR

11686 - Output Distributor
Defines which output devices are used when a default print 'PRINT A' is used.

11668 - Input Distributor
Defines which input device is used when a default input 'INPUT A$' is used.

Both distributor bytes are bit mapped according to the table shown below. Although more than one output device may be selected by setting multiple bits in the output distributor, the input distributor will only accept input from and device regardless of the number of bits set. Device scanning is from bit Ø to bit 7.

Bit	Dev	Physical Unit
Ø	1	Serial Console Port ($FCØØ)
1	2	Video Based Input & Display (44Ø/54Ø)
2	3	Serial Port(s) at $FBØØ
3	4	Memory I/O
4	5	Centronics Parallel (Output Only)
5	6	<Not Used-Word Processor Printer - Output Only>
6	7	<Not Used>
7	8	Serial Port(s) at $CFØØ

For example, to route normal output to the serial console and the line printer the commmand 'POKE 11686,17' should be used. This sets bit Ø and bit 4 which is a 1+16=17.

Memory I/O simply prints or inputs ASCII characters to RAM based on two 16 bit auto-incrememting memory pointers.

11657,11658 - Memory Input Pointer (Low/High)

11661,11662 - Memory Output Pointer (Low/High)

As characters are printed the memory output pointer will increment. The same applies for the memory input pointer on execution of an input. Before using memory I/O the program must poke these address pointers (Low/High) to point to the appropriate memory address.

Input or output may be directed to a specific device at any time through the use of the '#DV' addition to the 'INPUT' and 'PRINT' commmands. A 'PRINT#5' will print to the line printer and an 'INPUT#3' will input from device 3. In a time share environment the '#DV' approach is peferred to poking the I/O distributor because output device contention is automatically handled only if '#DV' output is used.

Function: USER PROGRAMMABLE CONTROL 'C'

Keywords: CONTROL C,INTERRUPT

The normal function of control 'c' is to stop the execution of the current program as if there had been a 'STOP' command executed. While the program is being run the operating system is constantly looking at the keyboard for a control 'c'. If control 'c' is turned off the system will not acknowledge a control 'c' but it still polls for it. This provides a means for interrupting the program under program control. The procedure is as follows:

1) Disable Control 'C' Acknowledgment

```
FLAG 25
```

2) Clear the Control 'C' Flag

```
POKE 15ØØ6,Ø
```

3) If Control 'C' is pressed then the contents of location 15ØØ6 will be non-zero.

4) To Restore Normal Control 'C' Operation

```
POKE 15ØØ6,Ø:REM Clear Flag
FLAG 26     :REM Enable Acknowledgement
```

There are several warnings that must be followed. First, control 'c' will not be polled when except on output to the console device. In order to interrupt one must occasionally output a non-printable character to the console, such as a 'CHR$(Ø);'. Second, remember to clear the flag BEFORE enabling control 'c'. If there is a control 'c' pending it will be acknowledged when control 'c' is enabled.

Function: SINGLE CHARACTER INPUT FROM THE CONSOLE

Keywords: INPUT, DETECT KEY

The 'INPUT' command in BASIC provides a line input capability. This means that the input will continue until a carriage return is entered. The routine below provides for a character input.

```
POKE 8778,135:POKE 8779,5
Z=USR(Z)
Z$=CHR$(PEEK(14518)AND127)
```

Where: Z$ = Character entered

Note: This routine will only work if the program 'EDITOR' Ver. 3.Ø or higher has been run or the DMS Plus Extension program 'INP$' has been run. The major advantage of this method is that it will work in all levels of 65U from single user through time-sharing and networking.

Function: PROGRAMMABLE PAGING ON OUTPUT DEVICES

Keywords: PAGING, PRINTER PAGING, CONSOLE, PARALLEL PORT

OS-65U provides for automatic paging on output devices one and five (#1 and #5). The program defines the paging based on the 'POKES' below.

Dev #5	Dev #1	Function
14387	15141	Total number of possible lines/page (Length * LPI)
14457	151ØØ	Number of printable lines/page (Total - top and bottom margins)
159Ø8	14358	Number of printable lines left on the current page

If no margins are required then the number of printable lines per page should be 'POKEd' with the 'PEEK' of the total number of possible lines per page. Right after bootup the paging for device one is set for no margins (i.e., paging off) while device five is set for eleven inch paper with three line margins at the top and bottom of the page. To disable paging on device five (#5) location 14457 should be 'poked' with the value in location 14387.

No other devices have paging registers; however, technical note 'TI1Ø18' covers the procedure for moving these registers to whatever port you wish.

Function: SERIAL PRINTER PAGING

Keywords: PAGING, SERIAL PRINTER, DEVICE 3, DEVICE 8

The POKE's below will add paging to device number 3 or device number 8. Print statements should be directed to device number 5. Note that the parallel printer driver is disabled when using these pokes. Two sets of POKEs are listed. The first set is for device number 8 ($CFØØ) and the second is for device number 3 ($FBØØ). Since multiple ports may reside on that device remember to poke the port index before output.

Device Number 8

```
POKE 15147,234:POKE 15148,234:POKE 15149,234
POKE 15879,Ø76:POKE 1588Ø,Ø91:POKE 15881,Ø77
POKE 159Ø2,234:POKE 159Ø3,234:POKE 159Ø4,234
POKE 19827,Ø76:POKE 19828,Ø25:POKE 19829,Ø62

Port Index - POKE 19798,((Port #)-1)*2
```

Device Number 3

```
POKE 15147,234:POKE 15148,234:POKE 15149,134
POKE 157Ø4,Ø76:POKE 157Ø5,Ø25:POKE 157Ø6,Ø62
POKE 15879,Ø76:POKE 1588Ø,Ø85:POKE 15881,Ø61
POKE 159Ø2,234:POKE 159Ø3,234:POKE 159Ø4,234

Port Index - POKE 1561Ø,((Port #)-1)*2
```

Function: CRT CONTROL CODES SUBROUTINE

Keywords: CRT INDEPENDENCE, CRT CODES

Application programs should be as peripheral independent as possible. Terminal control codes must be easy to change. One approach is to place them in a file which is read every time a program is run, but this forces additional disk transfers during the startup of the program. In a multi-user system this also forces all terminals to be of the same type with respect to control codes. Another approach is to inbed the codes in the operating system. If the Extended Input mode of 65U is active then a minimum subset of CRT codes are present in the operating system. They can be reterived with the following routine.

```
63900 Z=6345:AD=100:AD$="":DL$=""DE$="":AR=1:XF=0:YF=0
63904 Z1=PEEK(Z):REM - Address Cursor -
63905 IF Z1>127 THEN AR=2:Z1=Z1-128:REM Determine (x,y) Order
63906 AD$=AD$+CHR$(Z1)::REM Adr Cur Leadin
63907 Z=Z+1:Z1=PEEK(Z)
63908 IF Z1<128 AND Z1<>0 GOTO 63906
63909 IF Z1=0 GOTO 63915
63910 Z1=Z1-128
63911 DL$=DL$+CHR$(Z1):REM Adr Cur Delimiter
63912 Z=Z+1:Z1=PEEK(Z)
63913 IF Z<128 AND Z<>0 GOTO 63911
63914 IF Z1=0 GOTO 63915
63915 DE$="":GOTO 63917:REM Adr Cur Ending Delimter
63916 Z=Z+1:Z1=PEEK(Z)
63917 IF Z1<>0 THEN DE$=DE$+CHR$(Z1):GOTO 63916
63918 XF=PEEK(Z+1):YF=PEEK(Z+2):REM Adr Cur Offsets
63919 IF XF>127 THEN XF=XF-128:AR=AR+2:REM Binary/Ascii Flag
63920 Z=Z+3:CS$="":REM - CLr Scr -
63921 Z1=PEEK(Z):Z=Z+1:IF Z1<>0 THEN CS$=CS$+CHR$(Z1):GOTO 63921
63922 CS$=CS$+CHR$(13)
63923 CE$="":REM - Clr to End of Scr -
63924 Z1=PEEK(Z):Z=Z+1:IF Z1<>0 THEN CE$=CE$+CHR$(Z1):GOTO 63924
63925 CL$="":REM - Clr to End of Line -
63926 Z1=PEEK(Z):Z=Z+1:IF Z1<>0 THEN CL$=CL$+CHR$(Z1):GOTO 63926
63927 FG$="":REM - Foreground -
63928 Z1=PEEK(Z):Z=Z+1:IF Z1<>0 THEN FG$=FG$+CHR$(Z1):GOTO 63928
63929 BG$="":REM - Backgound -
63930 Z1=PEEK(Z):Z=Z+1:IF Z1<>0 THEN BG$=BG$+CHR$(Z1):GOTO 63930
63931 BL$=CHR$(7):RETURN
```

Once this subroutine is run the CRT functions listed below will work if the subroutine shown below is included in your program. The address the cursor subroutine will work on nearly all terminals including a DEC VT100.

```
           Address Cursor - Set (x,y) and 'GOSUB AD'
                            x - Horizontal Coord
                            y - Vertical Coord
                            (Ø,Ø) - Upper Left Corner of Screen

           Clear Screen   - 'PRINT CS$;'

   Clear to End of Screen - 'PRINT CE$;'

     Clear to End of Line - 'PRINT CL$;'

               Foreground - 'PRINT FG$;'

               Background - 'PRINT BG$;'

                Ring Bell - 'PRINT BL$;'

1ØØ POKE 22,X:ON AR GOTO 1Ø1,1Ø2,1Ø3,1Ø3:REM Address Cursor
1Ø1 PRINT AD$;CHR$(X+XF);DL$;CHR$(Y+YF);DE$;:RETURN
1Ø2 PRINT AD$;CHR$(X+XF);DL$;CHR$(X+XF);DE$;:RETURN
1Ø3 X$=MID$(STR$(X+1ØØ+XF):Y$=MID$(STR$(Y+1ØØ+YF)
1Ø4 IF AR=3 THEN PRINT AD$;X$;DL$;Y$;DE$;:RETURN
1Ø5 PRINT AD$;Y$;DL$;X$;DE$;:RETURN
```

The utility 'INP$' inserts the control codes into the operating system.

Function: USER PROGRAMMABLE ERROR RECOVERY

Keywords: ERROR TRAPPING, DISK ERRORS, BASIC ERRORS

As of version 1.3 of 65U the capability exists to send both disk errors and BASIC errors to line 5ØØØØ. The following routine is an example of how to decode them.

```
5ØØØØ EL=PEEK(11774)+256*PEEK(11775):REM Get Error Line
5ØØ1Ø EN=PEEK(18176):IF EN=23 GOTO 5Ø1ØØ:REM BASIC or Disk ?
5ØØ18 :
5ØØ19 REM Decode BASIC Error
5ØØ2Ø Z$=CHR$(PEEK(EN+867))+CHR$(PEEK(868+EN)):REM Error Code
5ØØ3Ø ER$="BASIC "+Z$+" Error in line"+STR$(EL)
5ØØ4Ø GOTO 5Ø2ØØ
5ØØ98 :
5ØØ99 REM Decode Disk Error
5Ø1ØØ EN=PEEK(1Ø226)
5Ø11Ø Z=PEEK(9832):IF Z>127 THEN Z=Z-124:IF Z>63 THEN Z=Z-58
5Ø12Ø ER$="Device "+CHR$(65+Z)" Disk Error"+STR$(EN)
5Ø13Ø ER$=ER$+" in line"+STR$(EL)
5Ø199 :
5Ø2ØØ PRINT:PRINT ER$:PRINT
5Ø21Ø END
```

To enable line 5ØØØØ error trapping a 'FLAG 23' must be executed. That will route both Disk and BASIC errors to line 5ØØØØ. If 'FLAG 9' is used then only Disk errors will go to line 5ØØØØ. BASIC errors will force the immediate mode.

Function: DISK FORMAT

Keywords: DISK FORMAT

A 65U floppy disk holds approximatly 275ØØØ characters of information. The floppy is divided into a system portion and a files portion. The system portion uses the first 25Ø87 bytes of the diskette. In this area a copy of the operating system is kept. When you 'boot' a diskette this is copied into the computer's memory and executed. Therefore if a diskette has not had a system copied to it, it will not boot. Since a program cannot use that area, it is recommended that a system always be copied to that area. (The program 'COPIER' does this.)

The files portion of the diskette starts at address 25Ø88 and extends to the end of the diskette. The first file is always the directory file (DIREC*). The operating system will not work if this file is not there. The directory can be as small as 3584 bytes or as large as 32768. 16 bytes per directory entry allows a maximum of 2Ø47 files per device. The second file on a floppy should be the program file 'BEXEC*'. This program is essential because it is always the first program to run when the operating system boots up.

To summarize; a floppy must have a system copied to it and also contain the file 'DIREC*' and the program 'BEXEC*' before it can be successfully booted.

Function: DIRECTORY ENTRY FORMAT

Keywords: DIRECTORY ENTRY, FILE HEADER

Each disk device under OS-65U has a directory. The directory has a file name 'DIREC*' and it always resides starting at 25088 bytes into the disk. The length is specified by the user at create time and can range between 3584 and 32768 bytes. Each file entry will use sixteen bytes of the directory so the length should be sixteen times the maximum number of files to be created on that device. The actual entry in the directory is an exact copy of the sixteen byte file header that resides at the beginning of each file. The format of this header is shown below:

```
         ________________________
 0 -   |                        |
 1 -   |                        |
 2 -   |       Filename         |
 3 -   |                        |
 4 -   |                        |
 5 -   |________________________|
 6 -   |       Password         |
 7 -   |________________________|
 8 -   |        Attribute       |
       |________________________|
 9 -   |                        |
10 -   |Disk Address  (Pages)   |
11 -   |________________________|
12 -   |                        |
13 -   | File Length  (Pages)   |
14 -   |________________________|
15 -   |     Special  *         |
       |________________________|
```

When a file is deleted a binary '1' is put in the first character position of the filename. The next open slot at the end of the directory is detected by the first character of the filename being a binary '0'. The file address and the file length are stored in binary from low byte to high in multiples of pages.

The last byte in each entry is a multi-purpose byte. DMS Plus Nucleus uses it to store the semaphore number of the file.

Function: CURRENT DISK DEVICE DETERMINATION

Keywords: CURRENT DEVICE

The current disk device letter can be determined by the following routine. It will work in all levels of OS-65U from single user to network configurations.

```
Z=PEEK(9832):IF Z>127 THEN Z=Z-124:IF Z>63 THEN Z=Z-58
DV$=CHR$(Z+65)

DV$ = Current Device Letter
       A,B,C,D - Floppy
       E       - Local Hard Disk
       K-Z     - Network Nodes
```

Function: DETERMINE THE DISK ADDRESS AND LENGTH OF A FILE

Keywords: DISK ADDRESS, FILE LENGTH

The absolute disk address and length of a file can be easily obtained by the following routine.

```
OPEN F$,P$,CH
Z=9898+CH*8
ADR=256*(PEEK(Z+1))+256*(PEEK(Z+2))+256*(PEEK(Z+3))
LN=256*(PEEK(Z+4))+256*(PEEK(Z+5))+256*(PEEK(Z+6))
```

```
Where:  F$ = Filename of the file you wish to locate
        P$ = Password
        CH = Channel
       ADR = Absolute Disk Starting Disk Address of
             the file 'F$'
        LN = Absolute Length of the file 'F$'
```

All files in 65U have a 16 byte file header so an index of zero is actually the sixteenth byte in the file. The variable 'LN' therefore is sixteen greater than the number of usable bytes and the beginning of the data is at a disk address of 'DA+16'.

Note: These PEEKs are only accurate immediately after opening the file.

Function: BLOCK DISK TRANSFERS FROM BASIC

Keywords: BASIC/DOS INTERFACE, RECORD R/W

Disk I/O under 65U is line oriented. There is a provision from the transfer to/from disk of larger contiguous blocks of memory. To do this only requires that four parameters be set up. They are the absolute disk address, the absolute RAM address, the number of bytes to transfer and whether to read or write to disk. The routine below does this.

```
1000 POKE 8778,192:POKE 8779,36:REM Set USR Vector to Interface
1010 POKE 9432,243:POKE 9433,40:REM Interface PUT Vector
1020 POKE 9435,232:POKE 9436,40:REM Interface GET Vector
1029 :
1030 CB=9889:REM Address of Disk Control Block
1039:
1040 DH=INT(DA/16777216):RM=DA-DH*16777216:REM Break up
1050 DM=INT(RM/65536)   :RM=RM-DM*65536   :REM  Address
1060 DL=INT(RM/256)     :RM=RM-DL*256     :REM   & Poke it
1069:
1070 POKE CB+1,RM:POKE CB+2,DL:POKE CB+3,DM:POKE CB+4,DH
1079 :
1080 POKE CB+5,NB-INT(NB/256)*256:REM Break up Bytes to
1090 POKE CB+6,INT(NB/256)       :REM  Transfer & Poke it
1099 :
1100 POKE CB+7,RA-INT(RA/256)*256:REM Break up RAM
1110 POKE CB+8,INT(RA/Q)         :REM  Address & Poke
1119 :
1120 DEV DV$:ER=USR(RW):REM Do Transfer
1129 :
1130 IF ER=0 GOTO 1150:REM Check for Error
1139 :
1140 PRINT "Error";ER;"at Address";DA
1149 :
1150 POKE 8778,208:POKE 8779,16:REM Set USR to FC Error
1160 END
```

DA	=	Disk Address
RA	=	RAM Address
NB	=	Number of Bytes to Transfer
RW	=	Read/Write Flag (0 or 1)
DV$	=	Device to do the Transfer on
ER	=	Error Code (0 if no error)

Although this method raises the throughput to the disk there is additional overhead that the program must bear. It is assumed that there is a buffer space defined somewhere in RAM. The disk address has to be determined. And finally and most importantly, this technique bypasses ALL filename logic. You must be very careful when calculating the disk address 'DA' to avoid destroying existing files.

Function: DATE & TIME DETERMINATION

Keywords: CURRENT DATE, CURRENT TIME

65U has provisions for maintaining a system date in the operating system. Under Time Share this is expanded to include the current time. For these locations to be accurate a set up program must poke them.

	Single User Intelligent Terminal	Time Share
Day	24569	55922
Month	2457Ø	55923
Year	24571	55924
Seconds	*	55919
Minutes	*	5592Ø
Hours	*	55921

* - Not Supported

Function: CURRENT OPERATING SYSTEM LEVEL DETERMINATION

Keywords: SYSTEM LEVEL,SYSTEM DETERMINATION

Version 1.3 of 65U provides an easy way of determining the system level. One byte has been reserved in the operating system for this purpose.

```
LV = PEEK(16317)

LV = System Level

     Ø - A release of 65U prior to version 1.3
     1 - Single User
     2 - Network Intelligent Terminal
     3 - Time Sharing
     4 - Time Sharing with Network Extension
```

WARNING: This location is set by the operating system and system programs. Applications programs should never poke into it!

Function: LEVEL III & NETWORK PEEKS

Keywords: LEVEL III, NETWORK, SEMAPHORE PEEK

The following memory locations can be used to determine level III and network system parameters. Any time an absolute memory location is referenced it should be done in a standard subroutine or a standard area of the program in order to minimize the changes needed if that location changes due to a new version of the operating system.

55381 - Time Share User Number (Ø-15)

57199 - Network Node Number (Ø=K,1=L,...,15=Z)

In a time share machine or on a network node the status of a semaphore can be obtained by the following routine.

```
Z=1:FOR Z1=1 TO SM-INT(SM/8)*8):Z=Z*2:NEXT Z1
A=1:IF (PEEK(55333+(SM/8)) AND Z) THEN A=Ø
RETURN
```

If semaphore 'SM' is set then a '1' is returned in 'A' else 'A' is equal to a 'Ø'

Function: SEMAPHORE TIMEOUT OPTION

Keywords: SEMAPHORE LOCKING, FILE CONTENTION

File contention in time sharing and network versions of 65U are handled through the use of semaphore flags. Normally if a program attempts to lock a semaphore that is already locked the system will suspend that program until the semaphore is available. In an interactive environment this may introduce unacceptable delays. The routine below permits the program to continue even if the semaphore was locked.

```
POKE 19632,TS
WAIT FOR SM
LC=2:IF PEEK(19633)<>Ø THEN LC=1
```

TS = # of seconds to wait for semaphore to unlock before returning. A delay value of '6Ø' will force a wait forever while a value from 'Ø' to '59' will force a wait for that number of seconds.

SM = Semaphore number to lock

LC = Status Indicator

1 - Program successfully locked semaphore
2 - Semaphore already locked by another user

Function: TIME SHARE OUTPUT DEVICE LOCKING

Keywords: TIME SHARE, OUTPUT DEVICE LOCKING

Level III provides for automatic output device locking on all output devices. If a user outputs to device 8 then no other user is allowed to print to that device until the user which set that device releases it by a 'PRINT#8!' or falls into the immediate mode. If multiple output units reside on a given device (e.g., CA-1ØX on device 8) then you may wish to disable this feature. The following table can be poked to set up the device locking mode. The values shown are the normal system defaults.

Hex	Dec	Dev	Contents
$DC66	56422	Ø	255
67	23	1	255
68	24	2	255
69	25	3	127
6A	26	4	255
6B	27	5	127
6C	28	6	127
6D	29	7	127
6E	3Ø	8	127
6F	31	9	127
7Ø	32	1Ø	127
71	33	11	127
72	34	12	127

255	Device is non-lockable
127	Device is lockable and not in use
N	Device is in use by user 'N' and locked

WARNING: If a device is made non-lockable then it is up to the program to handle device contention.

Function: MERGING OF BASIC PROGRAMS

Keywords: MERGING, SUBROUTINE

The procedure outlined below allows BASIC programs to be merged together without the hassle of indirect files.

LOAD"filename"	- Load the program that contains the code you wish to transfer.
OPEN"data file",1	- Open a scratch file. This file will hold the lines being transferred in an ASCII format so make sure that the file is bigger then the original BASIC file.
LIST%1,N-M	- List the lines you wish to transfer. If you generate an error then CLOSE and restart the procedure. You may use any form of the list statement. In the above example lines 'N' to 'M' have been listed to the data file.
PRINT%1,"OK"	- This is a end of data marker which is needed during the actual merge.
CLOSE 1	- Close the data file. Note: At this time the data file contains the lines listed in ASCII form as opposed to the normal tokenized form.

The above sequence has taken the lines of program code desired to be transferred and stored them in a transfer file. Now, to get the code into the program desired, the following procedure is necessary.

LOAD"filename"	- Load the file which is to receive the new lines.
OPEN"data file",1	- Open the file that contains the listed lines.
FLAG 13	- This flag permits a special form of the INPUT commmand to execute.
INPUT%1,	- This will input the lines from the data file until the end of data delimiter 'OK' is hit. This will cause a 'SN' error which will return control back to the operator.
CLOSE	- Close the data file.

FLAG 14	- Restore the INPUT flag.
CLEAR	- This CLEAR must be done.
SAVE"filename"	- Save the file which no has the new lines merged in.

Function: LIMITED FIND

Keywords: FIND, LIMITED FILE SEARCH

In normal operation the 'FIND' command searches from the current index to the end of file. The routine below will search for 'A$' from the current index to the index value equal to 'EI'. If no match was found the value of the index will be set to '1E9' just as in the unlimited find command.

```
1ØØØ Z=99Ø2+8*CH:Z1=PEEK(Z):Z2=PEEK(Z+1):Z3=PEEK(Z+2)
1Ø1Ø Z4=256:Z5=INT((EI+Z4)/Z4)*Z4
1Ø2Ø IF Z5 < EI+16 THEN Z5=Z5+Z4
1Ø3Ø IF Z5 => (Z1*Z4+Z2*Z4*Z4+Z3*Z4*Z4*Z4) GOTO 1Ø9Ø
1Ø4Ø POKE Z+2,INT(Z5/(Z4*Z4*Z4))
1Ø5Ø Z5=Z5-INT(Z5/(Z4*Z4*Z4))*(Z4*Z4*Z4)
1Ø6Ø POKE Z+1,INT(Z5/(Z4*Z4))
1Ø7Ø Z5=Z5-INT(Z5/(Z4*Z4))*(Z4*Z4)
1Ø8Ø POKE Z,INT(Z5/Z4)
1Ø9Ø FIND A$,CH
11ØØ POKE Z,Z1:POKE Z+1,Z2:POKE Z+2,Z3:RETURN
```

CH	= Channel Number
A$	= String to Search for (Ø-255 Characters)
EI	= Index for find to stop at if no find

This subroutine modifies the channel control block so if a disk error occurs during the find close and reopen the file before continuing.

Function: QUICK DISK BUFFER DUMP

Keywords: DISK BUFFER, FILE CONTENTION

Normally the single disk buffer is written back to the disk only when the buffer is needed for another sector and the contents of the buffer has changed since it was read from the disk. An example would be writing 1ØØ byte records sequentially to a disk file. The actual disk file would only be updated approximately every 35 records. (Note: The buffer size is 3584 bytes.) In many cases the programmer would like to force the operating system to write to the disk on a per record basis.

```
Z1=9898+CH*8
Z=PEEK(Z1):CLOSE CH:POKE Z1,Z
```

Where: CH = Channel Number (1-8)

This routine in effect 'closes' the channel specified, then 'opens' it again without the time-consuming disk read of the directory., Note: This routine will force the current buffer to disk. It will not force the buffer to be reloaded the next time it is accessed, therefore this routine alone cannot be used to implement record locking.

Function: DATA ONLY FLOPPY DISK CREATION

Keywords: DATA DISK FORMAT

Initialize a blank floppy diskette using the utility program 'COPIER'.

Copy a system from the utility diskette to the diskette that was initialized above.

Using the 'CREATE' utility to create a directory file on the data diskette with the characteristics of:

Name	- DIREC*
Length	- 3500
Type	- Other
Access Rights	- None
Password	- PASS

Using the 'CREATE' utility, create a file on the data diskette with the characteristics:

Name	- BEXEC*
Length	- 3500
Type	- BASIC
Access Rights	- RW

From the immediate mode clear the workspace with a 'NEW' command and type in the following program.

```
10 POKE 11686,1:POKE 11668,1
19 :
20 FOR X=1 TO 32:PRINT:NEXT
29 :
30 PRINT"Warning: This is a Data Diskette -- It should ";
32 PRINT"not be booted!"
39 :
40 GOTO 40
```

Save the program just typed in the newly created file 'BEXEC*' on the data diskette.

The data diskette is now ready to have data files created on it. If someone tries to boot up the diskette it will print a warning message since normally you should not be doing that in an application package.

Function: SETTING MEMORY SIZE

Keywords: MEMORY SIZE, ALLOCATION OF MEMORY

BASIC maintains a two byte address pointer (Low/High) which points to the current end of memory. These bytes were set by the system on boot up. To modify the size of the workspace one must poke the ending address into the locations:

```
POKE 132,AL:POKE 133,AH:CLEAR
```

Where AL and AH are the low and high parts of the address of the new end of the workspace. A 'CLEAR' must be executed for the change to take affect. After that the change will stay in effect until the system is rebooted.

Generating Machine Code

To assemble the machine code routines one should use the assembler under OS-65D V3.1 or the WP1-B word processor. The better choice is the WP1-B as it incorporates a full line EDITor, macro CHANGE and FIND commands, plus move and transfer commands. The machine code should be assembled to RUN at $6ØØØ and up. The assembled machine code must be Saved to the WP1-B or OS-65D V3.1 diskette. Since there exists no assembler under OS-65U and OS-65U can not read WP1B or OS-65D type diskettes, two utility programs have been provided. "LOAD32" and "LOAD48" (for 32K and 48K machines respectively) provide a means of calling machine code into OS-65U. When "RUN", the utilities "come up" in the OS-65D kernel mode i.e., the familar "A*" is output. At this point the machine code may be "called" into OS-65U. The steps below give the exact sequence.

1) Type:
   RUN "LOAD32", "PASS" <CR> for a 32K machine or for a 48K machine RUN "LOAD48", "PASS" <CR>

2) Home the floppy head by entering:
   A*Z

3) Call the machine code into place by entering:
   A*CXXXX=YY,Z
   A*Z

   Where XXXX is the address the machine code is to be called into (normaly $6ØØØ), YY is the track number and Z is the sector number.

4) Warm start OS-65U by entering:
   A*GXXXX

   Where XXXX=7E12 for a 32K machine and BE12 for a 48K machine.

5) Now enter:
   New XXXX <CR>

   Where XXXX stands for the number of bytes to be allocated for the machine code routines plus one.

   E.G. machine code runs from $6ØØØ to $6ØFF.
   The statement then would be
   "NEW 256",(($6ØFF+1)-$6ØØØ)= $61ØØ - $6ØØØ= $Ø1ØØ= 256
   (See diagrams on next page.)

6) Now enter the BASIC program to be used in conjunction with the machine code.

7) SAVE the BASIC program to disc.

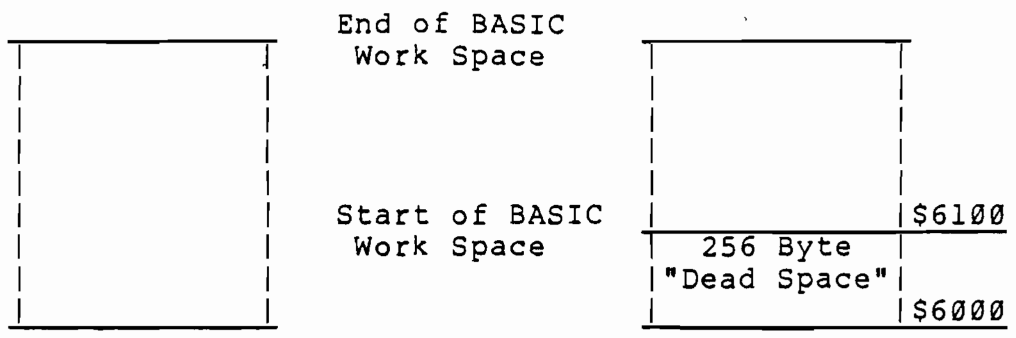
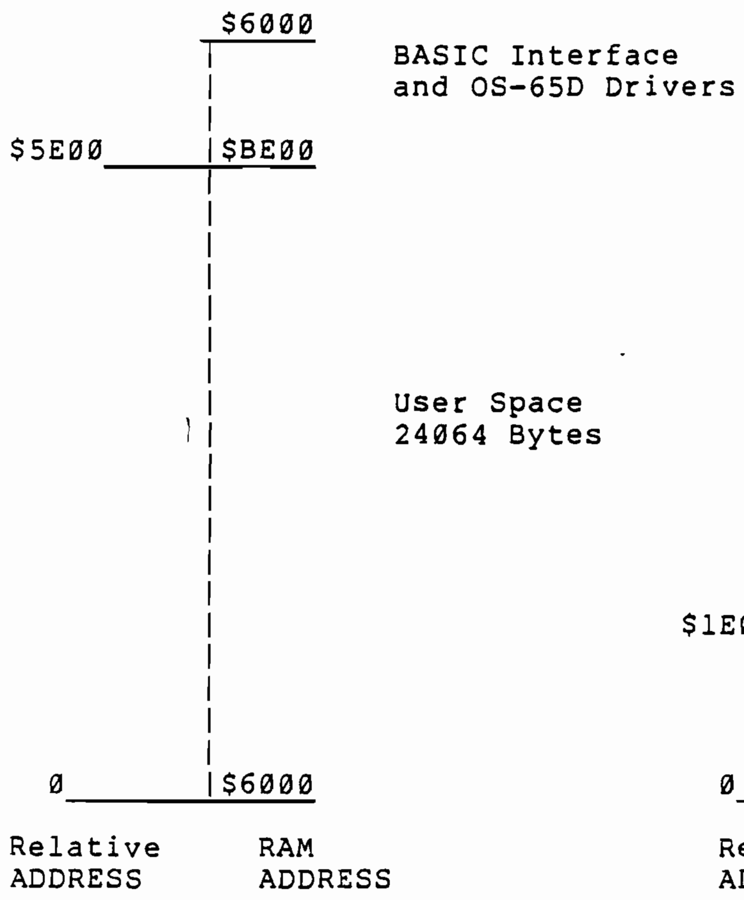
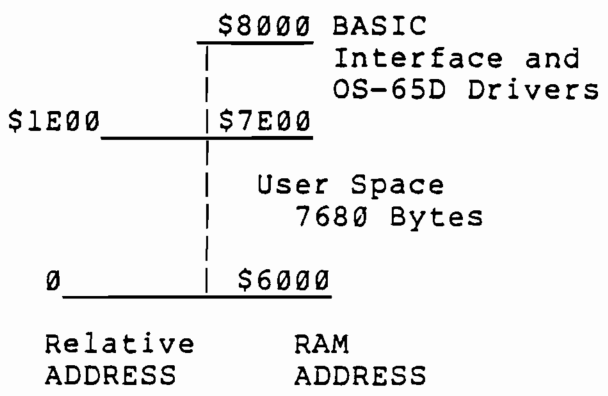


Diagram 1 Diagram 2



LOAD48



LOAD32

The diagrams above show how memory is allocated using "LOAD32" and "LOAD48".

System Error Codes

OS65U Floppy Disk Error Numbers

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	Old 65-D header found
27	Track 0 verification error
76	Track out of range

OS-65U CD-36 and CD-74 Hard Disk Error Numbers

1	Drive not ready
2	Seek 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	Cylinder mismatch
10	Track mismatch
11	Sector mismatch
16	Data field checksum error
24	Status error
25	Unit out of service
82	Cylinder number out of range

OS-65U CD-8(23) Hard Disk Error Numbers

```
 1 - Drive not ready
 2 - Seek timeout
 3 - Invalid unit number
 4 - Can't find cylinder zero
 5 - DMA failed to terminate
 6 - No data read
 7 - Sector unsafe
 8 - Header checksum
 9 - Cylinder mismatch
1Ø - Track mismatch
11 - Sector mismatch
12 - Data field checksum
13 - Status error
14 - Unit out of service
15 - Cylinder number out of range

+16 - In Position Head subroutine
+32 - In Seek Subroutine
+64 - In Select Unit Subroutine
+128- In Write Subroutine

EXAMPLE:  Error number  63
                       -32 Seek Subroutine
                        31
                       -16 Position Head
                        15    Subroutine

          Error is 15 - Cylinder Number Out of Range
```

OS-65U Network Error Numbers

```
238 - Output overrun
239 - Semaphore locked/not locked
24Ø - Relay response timeout
241 - Data Transmission Error : -- -- FE
242 - Data Transmission Error : -- OV
243 - Data Transmission Error : -- OV FE
244 - Data Transmission Error : PE -- --
245 - Data Transmission Error : PE -- FE
246 - Data Transmission Error : PE OV --
247 - Data Transmission Error : PE OV FE
      (Parity, Overrun, Framing)
248 - Control Block echo comparison
249 - Invalid initial poll command
25Ø - Data Block input timeout
251 - Control Block input timeout
252 - Incorrect poll code
253 - Incorrect poll responses
254 - Poll timeout
255 - Poll response timeout
```

Basic Language Error Messages

Error Code	Meaning
/Ø	Division by Zero.
BS	Bad Subscript: Index outside DIM statement range.
CN	Continue Error: Attempt to inappropriately continue. Can continue from BREAK or STOP if no lines changed or entered. Can't continue after any error.
DD	Double Dimension: Variable dimensioned twice. Remember subscripted variables default to dimension 1Ø.
FC	Function Call error: Parameter passed to function is out of range.
FS	Full Stack: Too many nested FORs or GOSUBs.
ID	Illegal Direct: INPUT and DEF statements cannot be used in direct mode.
LS	Long String: String too long.
NF	NEXT without FOR.
OD	Out of Data: More READs than DATA.
OM	Out of Memory: Program too big or too many variables.
OV	Overflow: Result of calculation too large.
RG	RETURN without GOSUB.
SN	Syntax error: Typo, etc.
SS	Semaphore stack overflow.
ST	String Temporaries: String expression too complex.
TM	Type Mismatch: String mismatched to numeric.
UF	Undefined Function: DEF must be executed before function is called.
US	Undefined Statement: Attempt to jump to non-existent line number.

OS-65U File System Error Codes

128	File not found
129	Channel not open
13Ø	Access Right violation
131	Executability violation
132	End of file
133	Channel already open

System Passwords

OS-65U System Passwords

Below are the passwords applicable to OS-65U V1.3. If desired, this page may be removed from the manual and stored separately for system security.

Utility Program Passwords

All utility programs with limited access (other than R/W) are assigned the password PASS, and must be RUN using this password.
Additional passwords are listed below.

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

The COPIER requires a password prior to initializing a hard disk DEVice.

The passwords are as follows:

Hard Disc type	Password
CD-74	33ØØ
CD-36	33ØØ
CD-23	4ØØØ
CD-7	1ØØØ

In BEXEC*, the password "UNLOCK" is needed to open the system for end user modifications.

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.

CRT Terminal Parameter Password

The password for the OS-DMS-PLUS data file "CRT Ø" is "PASS".