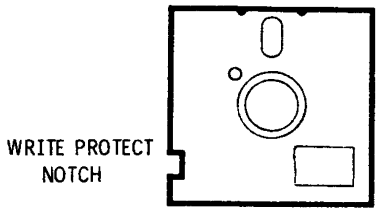
Notice to FLEX Users

Before experimenting with the FLEX operating system, it is a good idea to follow the steps given below to make a duplicate diskette in case you accidentally enter a command which would erase the supplied system diskette.

1.) Power up the computer system and disk system. Be sure that all memory is good and be sure that you have memory installed from hex ØØØØ- 2FFF (12K) and from hex 7ØØØ - 7FFF.

2.) Be sure that the small rectangular notch on the edge of the system diskette is **covered** with a small piece of tape. If not, cover it with a small piece of masking tape. This will write protect the diskette.



3.) Install the supplied system diskette in drive 0 (the left hand drive) as described in the disk manual and close the door.
(Write protect notch toward the bottom; end away from write protect notch inserted first.)

4.) Install a blank diskette in drive #1 and close the door. The write protect notch on this diskette should **not** be covered.

5.) Boot up the FDOS system as described in the manual by either entering the boot by hand or by typing D, depending on your monitor.

6.) The system should respond with FLEX and +++. If not, try to boot again as described in the manual. If after several trys the system cannot be booted, the system diskette should be removed and all hardware checked.

7.) When the system is booted type:
BACKUP Ø,1 followed by a carriage return.

8.) The system will then take several minutes to copy the disk. When the copying is finished, the system will respond: **BACKUP COMPLETE**.

9.) The supplied system diskette should now be removed and set aside. The copy can be tested and used as desired.

Advanced Programmer's Manual

Throughout this manual you will find references to the DOS Advanced Programmer's Guide. This manual contains detailed information on the operation of the Disk Operating System at the machine language level. It is written for the individual who wishes to write his own utilities, inter-face to the DOS thru machine language programs, or just understand how it all works. It has been written for the individual who understands programming at the machine language level and it is not recommended for the novice. It is not being supplied with the MF-68 kit but is sold separately for $20.00 ppd. in the continental U.S. It should be available sometime in May, 1978.

FLEX USER'S MANUAL

Technical Systems Consultants, Inc.
P.O. Box 2574
West Lafayette, Indiana 47906

COPYRIGHT NOTICE

PREFACE

The purpose of this User's Guide is to provide the user of the FLEX Operating System with the information required to make effective use of the available system commands and utilitites. This manual applies to the mini-floppy version of FLEX (sometimes referenced as "miniFLEX"). Most of the features of the larger versions of FLEX are fully operational on the mini version. The user should keep this manual close at hand while becoming familiar with the system. It is organized to make it convenient as a quick reference guide, as well as a thorough reference manual.

TABLE OF CONTENTS

I. INTRODUCTION

The FLEX® Operating System is a very versatile and flexible operating system. It provides the user with a powerful set of system commands to control all disk operations directly from the user's erminal. The systems programmer will be delighted with the wide variety of disk access and file management routines available for personal use. Overall, FLEX is one of the most powerful operating systems available today.

The FLEX Operating System is comprised of three parts, the File Management System (FMS), the Disk Operating System (DOS), and the Utility Command Set (UCS). Part of the power of the overall system lies in the fact that the system can be greatly expanded by simply adding additional utility commands. The user should expect to see many more utilities available for FLEX in the future. Some of the other important features include: fully dynamic file space allocation, the automatic "removal" of defective sectors from the disk, automatic space compression and expansion on all text files, complete user environment control using the TTYSET utility command, and uniform disk wear due to the high performance dynamic space allocator.

The UCS currently contains many very useful commands. These programs reside on the system disk and are only loaded into memory when needed. This means that the set of commands can be easily extended at any time, without the necessity of replacing the entire operating system. The utilities provided with FLEX perform such tasks as the saving, loading, copying, renaming, deleting, appending, and listing of disk files. There is an extensive CATalog command for examining the disk's file directory. Several environment control commands are also provided. Overall, FLEX provides all of the necessary tools for the user's interaction with the disk.

II. SYSTEM REQUIREMENTS

The minifloppy version of FLEX requires random access memory from location 0000 through location 2FFF hex (12K). Memory is also required from 7000 (28K) through 7FFF hex (32K), where the actual operating system resides. The system also assumes at least 2 disk drives are connected to the controller and that they are configured as drives #0 and #1. You should consult the disk drive instructions for this information. FLEX will work with either Motorola's MIKBUG® or Southwest Technical Products' SWTBUG® monitor ROMs. If using SWTBUG®,either the control interface (MP-C) or an ACIA-based serial interface (MP-S) will work. If using the serial Interface (MP-S), the 'escape' character (later defined) will stop any output to the terminal, and another 'escape' character will make it resume. This feature is not available on systems using the control interface (MP-C) with the monitor. One other note about systems containing a control interface (MP-C). It will be necessary to use the TTYSET utility command after system initialization to set the 'duplex' to 'half duplex mode' in order to suppress double printing of characters on the terminal. Consult the TTYSET description for details.

Ill. GETTING THE SYSTEM STARTED

Each FLEX system diskette contains a binary loader for loading the operating system into RAM. There needs to be some way of getting the loader off of the disk so it can do its work. This can be done by either hand entering the bootstrap loader provided with the disk system, or if SWTBUG® is installed in the system, simply type "D" to call the disk boot loader from ROM.

As a specific example, suppose the system we are using has SWTBUG® installed and we wish to run FLEX. The first step is to power on all equipment and make sure the SWTBUG® prompt is present ($). Next insert the system diskette into drive 0 (the boot must be performed with the disk in drive 0) and close the door on the drive. Type "D" on the terminal. The disk motors should start, and after about 2 seconds, the following should be displayed on the terminal:

```
FLEX X.X
+++
```

The name FLEX identifies the operating system and the X.X will be the version number of the operating system. The FLEX prompt is the three plus signs (+++), and will always be present when the system is ready to accept an operator command. The '+++' should become a familiar sight and signifies that FLEX is ready to work for you!

IV. DISK FILES AND THEIR NAMES

All disk files are stored in the form of 'sectors' on the disk and in the minifloppy version, each sector contains 128 'bytes' of information. Each byte can contain one character of text or one byte of binary machine information. A maximum of 612 sectors may be used on any one diskette, but the user need not keep count, for the system does this automatically. A file will always be a least one sector long and can have a maximum length of 612 sectors. The user should not be concerned with the actual placement of the files on the disk since this is done by the operating system. File deletion is also supported and all previously used sectors become immediately available again after a file has been deleted.

All files on the disk have a name. Names such as the following are typical:

PAYROLL
INVENTORY
TEST1234
APRIL-78
WKLY-PAY

Anytime a file is created, referenced, or deleted, its name must be used. Names can be most anything but must begin with a letter (not numbers or symbols) and be followed by at most 7 additional characters, called 'name characters'. These 'name characters' can be any combination of the letters 'A' through 'Z' or 'a' through 'z', any digit '0' through '9', or one of the two special characters, the hyphen (-) or the underscore _ (a left arrow on some terminals).

File names must also contain an 'extension'. The file extension further defines the file and usually indicates the type of information contained therein. Examples of extensions are: TXT for text type files, BIN for machine readable binary encoded files, CMD for utility command files, and BAS for BASIC source programs. Extensions may contain up to 3 'name characters' with the first character being a letter. Most of the FLEX commands assume a default extension on the file name and the user need not be concerned with the actual extension of the file. The user may at anytime assign new extensions, overiding the default value, and treat the extension as just part of the file name. Some examples of file names with their extension follow:

APPEND.CMD
LEDGER.BAS
TEST.BIN

Note that the extension is always separated from the name by a period '.'. The period is the name 'field separator'. It tells FLEX to treat the characters following the period as a new field in the name specification.

A file name can be further refined. The name and extension uniquely define a file on a particular drive, but the same name may exist on several drives simultaneously. To designate a particular drive a 'drive number' is added to the file specification. It consists of a single digit (0-3) and is separated from the name by the field separator '.'. The drive number may appear either before the name or after it (after the extension if it is given). If the drive number is not specified, the system will default to either the 'system' drive or the 'working' drive. These terms will be described a little later. Some examples of file specifications with drive numbers follow:

0.BASIC
MONDAY.2
1.TEST.BIN
LIST.CMD.1

In summary, a file specification may contain up to three fields separated by the field separator. These fields are: 'name', 'extension' and 'drive'. The rules for the file specification can be stated quite concisely using the following notation:

{(drive),}(name){.(extension)}
(name){.(extension)}{.(drive)}

The ‘( )’ enclose a field and do not actually appear in the specification, and the '{ }' surround optional items of the specification. The following are all syntactically correct:

0.NAME.EXT
NAME.EXT.0
NAME.EXT
0.NAME
NAME.0
NAME

Note that the only required field is the actual ‘name’ itself and the other values will usually default to predetermined values. Studying the above examples will clarify the notation used. The same notation will occur regularly throughout the manual.

V. ENTERING COMMANDS

When FLEX is displaying '+++', the system is ready to accept a command line. A command line is usually a name followed by certain parameters depending on the command being executed. There is no 'RUN' command in FLEX . The first file name on a command line is always loaded into memory and execution is attempted. If no extension is given with the file name, 'CMD' is the default. If an extension is specified, the one entered is the one used. Some examples of commands and how they would look on the terminal follow:

+++TTYSET
+++TTYSET.CMD
+++LOOKUP.BIN

The first two lines are identical to FLEX since the first would default to an extension of CMD. The third line would load the binary file 'LOOKUP.BIN' into memory and, assuming the file contained a transfer address, the program would be executed. A transfer address tells the program loader where to start the program executing after it has been loaded. If you try to load and execute a program in the above manner and no transfer address is present, the message, 'NO LINK' will be output to the terminal, where 'link' refers to the transfer address. Some other error messages which can occur are 'WHAT?' if an illegal file specification has been typed as the first part of a command line, and 'NOT THERE' if the file typed does not exist on the disk.

During the typing of a command line, the system simply accepts all characters until a 'RETURN' key is typed. Any time before typing the RETURN key, the user may use one of two special characters to correct any mistyped characters. One of these characters is the 'back space' and allows deletion of the previously typed character. Typing two back spaces will delete the previous two characters. The back space is initially defined to be a 'control H' but may be redefined by the user using the TTYSET utility command. The second special character is the line 'delete' character. Typing this character will effectively delete all of the characters which have been typed on the current line. A new prompt will be output to the terminal, but instead of the usual '+++' prompt, to show the action of the delete character, the prompt will be '???'. Any time the delete character is used, the new prompt will be '???', which signifies that the last line typed did not get entered into the computer. The delete character is initially a 'control X' but may also be redefined using TTYSET.

As mentioned earlier, the first name on a command line is always interpreted as a command. Following the command is an optional list of names and parameters, depending on the particular command being entered. The fields of a command line must be separated by either a space or a comma. The general format of a command line is:

(command){,(list of names and parameters)}

A comma is shown, but a space may be used. FLEX also allows several commands to be entered on one command line by use of the 'end of line' character. This character is initially a colon (':'), but may be user defined with the TTYSET utility. By ending a command with the end of line character, it is possible to follow it immediately with another command. FLEX will execute all commands on the line before returning with the '+++' prompt. An error in any of the command entries will cause the system to terminate operation of that command line and return with the prompt. Some examples of valid command lines follow:

+++CAT 1
+++CAT 1:ASN S=1
+++LIST LIBRARY:CAT 1:CAT 0

As many commands may be typed in one command line as desired, but the total number of characters typed must not exceed 128. Any excess characters will be ignored by FLEX.

One last system feature to be described is the idea of 'system' and 'working' drives. As stated earlier, if a file specification does not specifically designate a drive number, it will assume a default value. This default value will either be the current 'system' drive assignment or the current 'working' drive assignment. The system drive is the default for all command names, or in other words, all file names which are typed first on a command line . Any other file name on the command line will default to the working drive. When the system is first initialized, both the system drive and the working drive are set to drive 0. At this time, all drive defaults will be to drive 0. It is sometimes convenient to assign drive 1 as the working drive in which case all file references, except commands, will automatically look on drive 1. It is then convenient to have a diskette in drive 0 with all the system utility commands on it (the 'system drive'), and a disk with the files being worked on in drive 1 (the 'working drive'). If the system drive is 0 and the working drive is 1, and the command line was:

+++LIST TEXTFILE

FLEX would go to drive 0 for the command LIST and to drive 1 for the file TEXTFILE. The actual assignment of drives is performed by the ASN utility. See its description for details.

VI. COMMAND DESCRIPTIONS

There are two types of commands in FLEX, memory resident (those which actually are part of the operating system) and disk utility commands (those commands which reside on the disk and are part of the UCS). There are only two resident commands, GET and MON. They will be described here while the UCS (utility command set) is described in the following sections.

GET

The GET command is used to load a binary file into memory. It is a special purpose command and is not often used. It has the following syntax:

GET{,(file name list)}
where (file name list) is: (file spec){,(file spec)}etc.

Again the '{ }' surround optional items. 'File spec' denotes a file name as described earlier. The action of the GET command is to load the file or files specified in the list into memory for later use. If no extension is provided in the file spec, BIN is assumed. In other words, BIN is the default extension. Examples:

GET,TEST
GET,1.TEST,TEST2.O

where the first example will load the file named 'TEST.BIN' from the assigned working drive, and the second example will load TEST.BIN from drive 1 and TEST2.BIN from drive 0.

MON

MON is used to exit FLEX and return to the hardware monitor system such as SWTBUG®. The syntax for this command is simply MON followed by the 'RETURN' key.

NOTE: to re-enter FLEX after using the MON command, you should enter the program at location 7103 hex. If using MIKBUG® or SWTBUG®, simply typing 'G' will return you to the FLEX operating system.

The following pages describe all of the utility commands currently included in the UCS. You should note that the page numbers denote the first letter of the command name, as well as the number of the page for a particular command. For example, 'B.1.2' is the 2nd page of the description for the 1st utility name starting with the letter 'B'.

COMMON ERROR MESSAGES

Several error messages are common to many of the FLEX utility commands. These error messages and their meanings include the following:

NO SUCH FILE. This message indicates that a file referenced in a particular command was not found on the disk specified. Usually the wrong drive was specified (or defaulted), or a misspelling of the name was made.

ILLEGAL FILE NAME. This can happen if the name or extension did not start with a letter, or the name or extension field was too long (8 and 3 respectively). This message may also mean that the command being executed expected a file name to follow and one was not provided.

FILE EXISTS. This message will be output if you try to create a file with a name the same as one which currently exists on the same disk. Two files with the same name are not allowed to exist on the same disk.

SYNTAX ERROR . This means that the command line just typed does not follow the rules stated for the particular command used. Refer to the individual command descriptions for syntax rules.

GENERAL SYSTEM FEATURES

Any time one of the utility commands is sending output to the terminal, it may be temporarily halted by typing the 'escape' character (see TTYSET for the definition of this character). Once the output is stopped, the user has two choices: typing the 'escape' character again or typing 'RETURN'. If the 'escape' character is typed again, the output will resume. If the 'RETURN' is typed, control will return to FLEX and the command will be terminated. All other characters are ignored while output is stopped.

APPEND

The APPEND command is used to append or concatenate two or more files, creating a new file as the result. Any type of file may be appended but it only makes sense to append files of the same type in most cases. If appending binary files which have transfer addresses associated with them, the transfer address of the last file of the list will be the effective transfer address of the resultant file. All of the original files will be left intact.

DESCRIPTION

The general syntax for the APPEND command is as follows:

APPEND,(file spec){,(file list)},(file spec)

where (file list) can be an optional list of file specifications. It is necessary that the last file name specified does not exist on the disk since this will be the name of the resultant file. All other files specified must exist since they are the ones to be appended together. If only 2 file names are given, the first file will be copied to the second file. The extension default is TXT unless a different extension is used on the FIRST FILE SPECIFIED, in which case that extension becomes the default for the rest of the command line. Some examples will show its use:

```
APPEND,CHAPTER1,CHAPTER2,CHAPTER3,BOOK
APPEND,FILE1,1.FILE2.BAK,GOODFILE
```

The first line would create a file on the working drive called 'BOOK .TXT' which would contain the files 'CHAPTER1.TXT', CHAPTER2.TXT', and 'CHAPTER3.TXT' in that order. The second example would append 'FILE2.BAK' from drive 1 to FILE1.TXT from the working drive and put the result in a file called 'GOODFILE.TXT' on the working drive. The file GOODFILE defaults to the extension of TXT since it is the default extension. Again, after the use of the APPEND command, all of the original files will be intact, exactly as they were before the APPEND operation.

The ASN command is used for assigning the 'system' drive and the 'working' drive. The system drive is used by FLEX as the default for command names or, in general, the first name on a command line. The working drive is used by FLEX as the default on all other file specifications within a command line. As the system is initialized, both the system and working drives are set to drive 0. An example will show how the system defaults to these values:

```
APPEND,FILE1,FILE2,FILE3
```

If the system drive is Ø and the working drive is assigned to drive 1, then the above example will perform the following operation: get the APPEND command from drive Ø (the system drive), then append FILE2 from drive 1 (the working drive) to FILE1 from drive 1 and put the result in FILE3 on drive 1. As can be seen, the system drive was the default for APPEND where the working drive was the default for all other file specs listed.

DESCRIPTION

The general syntax for the ASN command is as follows:

```
ASN{,W=(drive)}{,S=(drive)}
```

where (drive) is a single digit drive number. If just ASN is typed followed by a 'RETURN', no values will be changed, but the system will output a message which tells the current assignments of the system and working drives, for example:

```
+++ASN
THE SYSTEM DRIVE IS #Ø
THE WORKING DRIVE IS #Ø
```

Some examples of using the ASN command are:

```
ASN,W=1
ASN,S=1,W=Ø
```

Where the first line would set the working drive to 1 and leave the system drive assigned to its previous value. The second example sets the system drive to 1 and the working drive to Ø. Careful use of drive assignments will allow the operator to avoid the use of drive numbers on file specifications most of the time!

BACKUP

The BACKUP command allows for the making of copies of entire FLEX disks. These copies are different from those produced by the COPY command in that BACKUP makes a "mirror image" copy of the input disk, where COPY always reorganizes a disk so that a file's sectors are all grouped together. There are trade-offs involved when deciding whether to use the BACKUP command or the COPY command. Reorganization will speed up file accesses which have become slow due to the sectors of a file not being grouped together. Generally, COPY should be used if there are only a few files on the disk, or if the disk is very slow in access times. COPY will also allow single files to be copied as well as copying files to partially used sides. The BACKUP command, which in most cases will run faster than the COPY routine, will only copy entire disks, and the output disk will be entirely overwritten. Experience will help determine which command to use and when.

DESCRIPTION

The general syntax for the BACKUP command is:

BACKUP,(input drive),(output drive)

where the drives are specified with single digits. 'The input drive contains 'the disk we wish copy the information from, and the output drive contains the disk on which we wish the data to be placed. As an example, to BACKUP drive Ø to drive1, the following should be typed:

+++BACKUP,Ø,1

There are several situations which can exist at the start of a BACKUP operation. Since the BACKUP command copies every sector from the input drive to the output drive, not caring if there is actually information on those sectors, it requires that the output disk have no bad sectors. The first thing BACKUP does is to check the output disk to make sure there are no bad sectors and also checks if the disk has been initialized (by use of the NEWDISK command). If there are bad sectors, the backup process will be aborted and a message to that effect will be ouput to the terminal. If the disk has not been initialized, the BACKUP routine will automatically perform the formatting process, again checking for bad sectors. If the output disk is not perfectly "clean" (either new without initialization or freshly formatted, in other words, the disk has files on it), the BACKUP command will issue the following message to the terminal:

SCRATCH DISK NOT BLANK
ARE YOU SURE?

If you still want to continue with the BACKUP procedure, type 'Y' as a response, otherwise type 'N' and BACKUP will be aborted.

One final note will be of interest. If the input disk had DOS.SYS on it, and it had been previously linked to the boot (see LINK command), then the new disk will also have DOS.SYS and it will be linked to the boot as well.

BUILD

The BUILD command is provided for those desiring to create small text files quickly (such as STARTUP files, see STARTUP) or not wishing to use the optionally available FLEX Text Editing System. The main purpose for BUILD is to generate short text files for use by either the EXEC command or the STARTUP facility provided in FLEX.

DESCRIPTION

The general syntax of the BUILD command is:

BUILD,(file spec)

where (file spec) is the name of the file you wish to be created. The default extension for the spec is TXT and the drive defaults to the working drive. The ouput file must not already exist or the error message, 'FILE EXISTS' will be issued.

After you are in the 'BUILD' mode, the terminal will respond with the equals sign ('=') as the prompt character This is similar to the Text Editing System's prompt for text input. To enter your text, simply type on the terminal the desired characters, keeping in mind that once the 'RETURN' is typed, the line is in the file and can not be changed. Any time before the 'RETURN' is typed, the backspace character may be used as well as the line delete character. If the delete character is used, the prompt will be '???' instead of the equals sign to show that the last line was deleted and not entered into the file. It should be noted that only printable characters (not control characters) may be entered into text files using the BUILD command.

To exit the BUILD mode, it is necessary to type a pound sign ('#') immediately following the prompt, then type 'RETURN'. The file will be finished and control returned back to FLEX where the three plus signs should again be output to the terminal. This exiting is similar to that of the Text Editing System.

CAT

The CATalog command is used to display the FLEX disk file names in the directory on each disk. The user may display selected files on one or multiple drives if desired.

DESCRIPTION

The general syntax of the CAT command is:

CAT{,(drive list)}{,(match list)}

where (drive list) can be one or more drive numbers separated by commas, and (match list) is a set of name and extension characters to be matched against names in the directory. For example, if only file names which started with the characters 'VE' were to be cataloged, then VE would be in the match list. If only files whose extensions were 'TXT' were to be cataloged, then TXT should appear in the match list. A few specific examples will help clarify the syntax:

```
+++CAT
+++CAT,1,A.T,DR
+++CAT,PR
+++CAT,Ø,1
+++CAT,Ø,1,.CMD,.SYS
```

The first example will catalog all file names on the working drive. The second example will catalog only those files on drive 1 whose names begin with 'A' and whose extensions begin with 'T', and also all files on drive 1 whose names start with 'DR'. The next example will catalog all files on the working drive whose names start with 'PR'. The next line causes all files on both drive Ø and drive 1 to be cataloged. Finally, the last example will catalog the files on drive Ø and 1 whose extensions are CMD or SYS.

During the catalog operation, before each drive's files are displayed, a header message stating the drive number is output to the terminal. The actual directory entries are listed in the following form:

NAME.EXTENSION size

where size is the number of sectors that file occupies on the disk. If more than one set of matching characters was specified on the command line, each set of names will be grouped according to the characters they match. For example, if all .TXT and .CMD files were cataloged, the TXT types would be listed together, followed by the CMD types.

In summary, if the CAT command is not parameterized, then all files on the assigned working drive will be displayed. If it is parameterized by only a drive number, then all files on that drive will be displayed. If the CAT command is parameterized by only an extension, then only files with that extension will be displayed. If only the name is used, then only files which start with that name will be displayed. If the CAT command is parameterized by only name and extension, then only files of that root name and root extension (on the working drive) will be displayed. Learn to use the CAT command and all of its features and your work with the disk will become a little easier.

COPY

The COPY command is used for making copies of files on a disk. Individual files, groups of name-similar files, or entire disks may be copied. The COPY command is a very versatile utility. The COPY command also re-groups the sectors of a file in case they were spread all over the old disk. This regrouping can make file access times much faster. When copying entire disks it is sometimes more desirable to use the BACKUP command. Refer to its description for details of the tradeoffs involved between the two methods of copying disks. It should be noted that before copying files to a new disk, the disk must be formatted first. Refer to NEWDISK for instructions on this procedure.

DESCRIPTION

The general syntax of the COPY command has three forms:

a . COPY,(file spec),(file spec)
b . COPY,(file spec),(drive)
c . COPY,(drive),(drive){,(match list)}

where match list) is the same as that described in the CAT command and all rules apply to matching names and extensions. When copying files, if the destination disk already contains a file with the same name as the one being copied, the file name and the message: FILE EXISTS DELETE ORIGINAL? will be output on the terminal. Typing Y will cause the file on the destination disk to be deleted and the file from the source disk will be copied to the destination disk. Typing N will direct FLEX not to copy the file in question.

The first type of COPY allows copying a single file into another. The output file may be on a different drive but if on the same drive, the file names must be different. It is always necessary to specify the extension of the input file but the output file's extension will default to that of the input's if none is specified. An example of this form of COPY is:

```
+++COPY,Ø.TEST.TXT,1.TEXT25
```

This command line would cause the file TEST.TXT on drive Ø to be copied into a file called TEST25.TXT on drive 1. Note how the second file's extension defaulted to TXT, the extension of the input file.

The second type of COPY allows copying a file from one drive to another drive with the file keeping its original name. An example of this is:

```
+++COPY,Ø.LIST.CMD,1
```

Here the file named LIST.CMD on drive Ø would be copied to drive 1. It is again necessary to specify the file's extension in the file specification. This form of the command is more convenient than the previous form if the file is to retain its original name after the copying process.

The final form of COPY is the most versatile and the most powerful. It is possible to copy all files from one drive to another, or to copy only those files which match the match list characters given. Some examples will clarify its use:

```
+++COPY,Ø,1
+++COPY,1,Ø.CMD,.SYS
+++COPY,Ø,1,A,B,CA.T
```

The first example will copy all files from drive Ø to drive 1 keeping the same names in the process. The second example will copy only those files on drive 1 whose extensions are CMD and SYS to drive Ø. No other files will be copied. The last example will copy the files from drive Ø whose names start with 'A' or 'B' regardless of extension, and those files whose names start with the letters 'CA' and whose extensions start with 'T', to the ouput drive which is drive 1. The last form of copy is the most versatile because it will allow putting just the command (CMD) files on a new disk, or just the SYS files, etc., with a single command entry. During the COPY process, the name of the file which is currently being copied will be ouput to the terminal, as well as the drive to which it is being copied.

COPYNEW

The COPYNEW command is similar to the COPY command but is normally used for copying only files from a source disk which do not exist on the destination diskette. This gives the user the ability to update a disk with all new files which do not already exist on it.

DESCRIPTION

The general syntax of the COPYNEW command is of the form:

COPYNEW,(drive),(drive){,(match list)}

where (match list) is the same as that described in the CAT command and all rules apply to matching names and extensions. When the COPYNEW command is used an attempt will be made to copy all files from the source disk to the destination disk. If the file name does not exist on the destination disk the file will be copied. If the file name does exist on the destination disk, the message FILE EXISTS will be displayed arid that file will not be copied. COPYNEW does not give you the option of deleting the old file on the destination disk like COPY does.

Some examples of COPYNEW are as follows:

+++COPYNEW Ø,1

+++COPYNEW Ø,1,C,.BAS

The first example will copy all files from drive Ø to drive 1 which do not already exist on drive 1. The second example will copy all files from drive Ø that begin with C or that have the extension .BAS to drive 1 provided that Ø. The file does not already exist on drive 1.

DELETE

The DELETE command is used to delete a file from the disk. Its name will be removed from the directory and its sector space will be returned to the free space on the disk.

DESCRIPTION

The general syntax of the DELETE command is:

DELETE,(file spec){,(file list)}

where (file list) can be an optional list of file specifications. It is necessary to include the extension on each file specified. As the DELETE command is executing it will prompt you with:

DELETE "FILE NAME"?

The entire file specification will be displayed, including the drive number. If you decide the file should be deleted, type 'Y', otherwise, any other response will cause that file to remain on the disk. If a 'Y' was typed, the message 'ARE YOU SURE?' will be displayed on the terminal. If you are absolutely sure you want the file deleted from the disk, type another 'Y' and it will be gone. Any other character will leave the file intact. ONCE A FILE HAS BEEN DELETED, THERE IS NO WAY TO GET IT BACK! Be absolutely sure you have the right file before answering the prompt questions with Y's. Once the file is deleted, the space it had occupied on the disk is returned back to the list of free space for furture use by other files. Few examples follow:

+++DELETE,MATHPACK.BIN

+++DELETE,1.TEST.TXT,Ø.AUGUST.TXT

The first example will DELETE the file named MATHPACK.BIN from the working drive. The second line will DELETE the file TEST.TXT from drive 1, and AUGUST.TXT from drive Ø.

The EXECute command is used to process a text file as a list of commands, just as if they had been typed from the keyboard. This is a very powerful feature of FLEX for it allows very complex procedures to be built up as a command file. When it is desirable to run this procedure, it is only necessary to type EXEC followed by the name of the command file. Essentially all EXEC does is to replace the FLEX keyboard entry routine with a routine which reads a line from the command file each time the keyboard routine would have been called. The FLEX utilities have no idea that the line of input is coming from a file instead of the terminal.

DESCRIPTION

The general syntax of the EXEC command is:

```
EXEC,(file spec)
```

where (file spec) is the name of the command file. The default extension is TXT. An example will give some ideas on how EXEC can be used. One set of commands which might be performed quite often is the set to make a new system diskette on drive 1 (see NEWDISK). Normally it is necessary to use NEWDISK and then copy all .CMD and all .SYS files to the new disk. Finally the LINK must be performed. Rather than having to type this set of commands each time it was desired to produce a new system diskette, we could create a command file called MAKEDISK.TXT which contained the necessary commands. The BUILD utility should be used to create this file. The creation of this file might go as follows:

```
+++BUILD,MAKEDISK
  =NEWDISK,1
  =COPY,Ø,1,.CMD,.OV,.LOW,.SYS
  =LINK,1.DOS
  =*
+++
```

The first line of the example tells FLEX we wish to BUILD a file called MAKEDISK (with the default extension of .TXT). Next, the three necessary command lines are typed in just as they would be typed into FLEX. The COPY command will copy all files with CMD, OV, LOW, and SYS extensions from drive Ø to drive 1. Finally the LINK will be performed. Now when we want to create a system disk in drive 1 we only need to type the following:

```
+++EXEC,MAKEDISK
```

We are assuming here that MAKEDISK resides on the same disk which contains the system commands. EXEC can also be used to execute the STARTUP file (see STARTUP).

There are many applications for the EXEC command. The one shown is certainly useful but experience and imagination will lead you to other useful applications.

JUMP

The JUMP command is provided for convenience. It is used to start execution of a program already stored in computer RAM memory.

DESCRIPTION

The general syntax of the JUMP command is:

JUMP,(hex address)

where (hex address) is a 1 to 4 digit hex number representing the address where program execution should begin. The primary reason for using JUMP is if there is a long program already in memory and you do not wish to toad it off of the disk again. Some time can be saved but you must be sure the program really exists before JUMPing to it!

As an example, suppose we had a BASIC interpreter in memory and it had a 'warm start' address of 1Ø3 hex. To start its execution from FLEX, type the following:

+++JUMP,1Ø3

The BASIC interpreter would then be executed. Again, remember that you must be absolutely sure the program you are JUMPing to is actually present in memory.

LINK

The LINK command is used to tell the bootstrap loader where the DOS.SYS file resides on the disk. This is necessary each time a system disk is created using NEWDISK. The NEWDISK utility should be consulted for complete details on the use of LINK.

DESCRIPTION

The general syntax of the LINK command is:

```
LINK,(file spec)
```

where (file spec) is usually DOS. The default extension is SYS. Some examples of the use of LINK follow:

```
+++LINK,DOS
+++LINK,1.DOS
```

The first line will LINK DOS.SYS on the working drive, while the second example will LINK DOS.SYS on drive 1. For more advanced details of the LINK utility, consult the "Advanced Programmers Guide".

LIST

The LIST command is used to LIST the contents of text or BASIC files on the terminal. It is often desirable to examine a file without having to use an editor or other such program. The LIST utility allows examining entire files, or selected lines of the file. Line numbers may also be optionally printed with each line.

DESCRIPTION

The general syntax of the LIST command is:

LIST,(file spec){,(line range)}{,N}

where the (file spec) designates the file to be LISTed (with a default extension of TXT), (line range) is the first and last line number of the file which you wish to be displayed, and the 'N' tells LIST to output line numbers. All lines are output if no range specification is given. A few examples will clarify the syntax used:

```
+++LIST,RECEIPTS,N
+++LIST,CHAPTER1,3Ø-2ØØ,N
+++LIST,LETTER,1ØØ
```

The first example will list the file named 'RECEIPTS.TXT' with line numbers. All lines will be output unless the escape 'character' is used as described in the Utility Command Set introduction. The second example will LIST the 3Øth line through the 2ØØth line of the file named 'CHAPTER1. TXT' on the terminal. The hyphen ('-') is required as the range number separator. Line numbers will be output because of the 'N'. The last example shows a special feature of the range specification If only one number is stated, it will be interpreted as the first line to be displayed. All lines following that line will also be LISTed. The last example will LIST the lines from line 1ØØ to the end of the file. No line numbers will be output since the 'N' was omitted.

MEMTEST1

The MEMTEST1 utility can be used to verify the integrity of the computer's memory. MEMTEST1 should be run periodically on your computer to alert you of any memory failures.

DESCRIPTION

The general syntax of the MEMTEST1 utility is:

MEMTEST1

MEMTEST1 does not have any arguments or file specifications associated with it. MEMTEST1 will then prompt you for the beginning and ending memory addresses. A four digit hexadecimal number should be entered in each case. In the case of a 32K system, the response would be as follows:

```
+++MEMTEST1
ENTER THE STARTING MEMORY ADDRESS ØØØØ
ENTER THE ENDING MEMORY ADDRESS 7FFF
```

If no errors are found in the memory being checked a + will be displayed on the screen. To completely test an area of memory, MEMTEST1 must be allowed to run until 256 +'s have been displayed on the screen. Each time a + is displayed on the screen MEMTEST has successfully cycled through memory storing and reading a different pattern.

If an error is detected the output will be similar to the following:

```
$Ø6        2Ø        16AØ
(PATTERN #)  (ERRANT BITS)  (ADDRESS)
```

An error message such as this says that MEMTEST1 cycled thru memory five times without error, but on the sixth try a pattern was used that detected an error. The Ø6 tells what pattern number MEMTEST1 was working on when the error was detected. The 2Ø (hexadecimal) tells which bit(s) were in error. 2Ø converted to binary is ØØ1ØØØØØ-the location of the 1 is the bit(s) that were in error, in this case bit 5. Bit numbers start from Ø as shown.

```
          7 6 5 4 3 2 1 Ø BIT #
20₁₆ = Ø Ø 1 Ø Ø Ø Ø Ø
```

The 16AØ is the address where the error was detected. This address may not store a particular number or possibly writing into another address, such as 16BØ, changed the contents of 16AØ.

The IC assignments table supplied with the memory board should be used to help locate the problem. In the above case on an MP-8M 8K memory board the bit # 5 IC in the upper 4K of memory should be suspected.

After running MEMTEST1, FLEX may be re-entered only by re-booting the system.

NEWDISK

NEWDISK is used to format a new diskette. Diskettes as purchased will not work with FLEX until certain system information has been put on them. The NEWDISK utility puts this information on the diskette, as well as checking the diskette for defective sectors (bad spots on the surface of the disk which may cause data errors).

DESCRIPTION

The general syntax of the NEWDISK command is:

NEWDISK,(drive)

where (drive) represents a single digit drive number and specifies the drive to be formatted. After typing the command, the system will ask if you are sure you want a NEWDISK, and if the disk to be initialized is a scratch disk. Type 'Y' as the response to these questions if you are sure the NEWDISK command should continue.

The NEWDISK process takes approximately one and one-half minutes to initialize a disk, assuming there are no bad spots on it. Defective sectors will make NEWDISK run even slower, depending on the number of bad sectors found. As bad sectors are detected, a message will be output to the terminal such as:

BAD SECTOR AT xx yy

where "xx" is the disk track number (in hex) and 'yy' is the sector number, also in hex. NEWDISK automatically removes bad sectors from the list of available sectors, so even if a disk has several bad sectors on it, it is still usable. When NEWDISK finishes, FLEX will report the number of available sectors remaining on the disk. If no defective sectors were detected, the total should be 612.

Sometimes during the NEWDISK process, a sector will be found defective in an area on the disk which is required by the operating system. In such a case, NEWDISK will report:

FATAL ERROR-FORMATTING ABORTED

and FLEX will regain control. You should not immediately assume the disk to be useless if this occurs, but instead, remove the disk from the drive, re-insert it, and try NEWDISK again. If after several attempts the formatting is still aborted, you should assume the disk is unusable. You may not BACKUP onto a diskette with bad sectors on it. See the BACKUP documentation for more information.

CREATING SYSTEM DISKETTES

A system disk is one which the disk operating system can be loaded from. Normally the system disk will also contain the Utility Command Set (UCS). The following procedure should be used when preparing system disks.

1. Initialize the diskette using NEWDISK as described above.
2. COPY all .CMD files desired to the new disk.
3. COPY all .SYS files to the new disk. It should be noted that steps 2 and 3 can be done with one command; 'COPY,Ø,1,.CMD,.OV,.LOW,.SYS', assuming you are copying from Ø to 1 and all command files and their overlays are desired. (the .OV copies overlay files and .LOW copies the utility 'SAVE.LOW').
4. Last it is necessary to LINK the file DOS.SYS to the system using the LINK command.

A very convenient way to get the above process performed without having to type all of the commands each time is to create a command file and use the EXEC command. Consult the EXEC documentation for details.

It is not necessary to make every disk a system diskette. It is also possible to create 'working' diskettes, disks which do not have the operating system on them, for use with text files or BASIC files. Remember that a diskette can not be used for booting the system unless the operating system is contained on it. To create a working disk, simply run NEWDISK on a diskette. It will now have all of the required information to enable FLEX to make use of it. This disk, however, does not contain the disk operating system and is not capable of booting the system.

The P command is very special and unlike any others currently in the UCS. P is the system print routine and will allow the output of any command to be routed to the printer. This is very useful for getting printed copies of the CATalog or when used with the LIST command will allow the printing of FLEX text files.

DESCRIPTION

The general syntax of the P command is:

P, (command)

where (command) can be any standard utility command line. If P is used with multiple commands per line (using the 'end of line' character,:), it will only have affect on the command it immediately precedes. Some examples will clarify its use:

+++P,CAT

+++P,LIST,MONDAY:CAT,1

The first example would print a CATalog of the directory of the working drive on the printer. The second example will print a LISTing of the text file MONDAY.TXT and then display on the terminal a CATalog of drive 1 (this assumes the 'end of line' character is a':'). Note how the P did not cause the 'CAT,1' to go to the printer. Consult the 'Advanced Programmer's Guide' for details concerning adaption of the P command to various printers.

The P command tries to load a file named PRINT.SYS from the same disk which P itself was retrieved. The PRINT.SYS file which is supplied with the system diskette contains the necessary routines to operate a SWTPC PR 4Ø printer connected through a parallel interface on PORT 7 of the computer. If you wish to use a different printer configuration, consult the 'Advanced Programmer's Guide' for details on writing your own printer driver routines to replace the PRINT.SYS file. The PR 4Ø drivers, however, are compatible with many other parallel interfaced printers presently on the market.

RENAME

The RENAME command is used to give an existing file a new name in the directory. It is useful for changing the actual name as well as changing the extension type.

DESCRIPTION

The general syntax of the RENAME command is:

RENAME,(file spec 1),(file spec 2)

where (file spec 1) is the name of the file you wish to RENAME and (file spec 2) is the new name you are assigning to it. The default extension for file spec 1 is TXT and the default drive is the working drive. If no extension is given on (file spec 2), it defaults to that of (file spec 1). No drive is required on the second file name, and if one is given it is ignored. Some examples follow:

```
+++RENAME,TEST1.BIN,TEST2
+++RENAME,1.LETTER,REPLY
+++RENAME,Ø.FIND.BIN,FIND.CMD
```

The first example will RENAME TEST1.BIN to TEST2.BIN. The next example RENAMEs the file LETTER.TXT on drive 1 to REPLY.TXT. The last line would cause the file FIND.BIN on drive Ø to be renamed FIND.CMD. This is useful for making binary files created by an assembler into command files (changing the extension from BIN to CMD). If you try to give a file a name which already exists in the directory, the message:

FILE EXISTS

will be displayed on the terminal. Keep in mind that RENAME only changes the file's name and in no way changes the actual file's contents.

One last note of interest. Since utility commands are just like any other file, it is possible to rename them also. If you would prefer some of the command names to be shorter, or different all together, simply use RENAME and assign them the names you desire.

SAVE

The SAVE command is used for saving a section of memory on the disk. Its primary use is for saving programs which have been loaded into memory from tape or by hand.

DESCRIPTION

The general syntax of the SAVE command is:

SAVE,(file spec),(begin adr),(end adr){,(transfer adr)}

where (file spec) is the name to be assigned to the file. The default extension is BIN and the default drive is the working drive. The address fields define the beginning and ending addresses of the section of memory to be written on the disk. The addresses should be expressed as hex numbers. The optional (transfer address)would be included if the program is to be loaded and executed by FLEX This address tells FLEX where execution should begin. Some examples will clarify the use of SAVE:

+++SAVE,DATA,1ØØ,1FF
+++SAVE,1.GAME,Ø,168Ø,1ØØ

The first line would SAVE the memory locations 1ØØ to 1FF hex on the disk in a file called DATA.BIN. The file would be put on the working drive and no transfer address would be assigned. The second example would cause the contents of memory locations Ø through 168Ø to be SAVEd on the disk in file GAME.BIN on drive 1. Since a transfer address of 1ØØ was specified as a parameter, typing 'GAME.BIN' in response to the FLEX prompt after saving would cause the file to be loaded back into memory and execution started at location 1ØØ.

It is not possible to SAVE a file under a name which already exists on the disk. In other words, the name you are assigning to the file being SAVEd must not already exist on the disk or the error 'FILE EXISTS' will be displayed.

Sometimes it is desirable to save noncontiguous segments of memory. To do this it would be necessary to first SAVE each segment as a separate file and then use the APPEND command to combine them into one file. If the final file is to have a transfer address, you should assign it to one of the segments as it is being SAVEd. After the APPEND operation, the final file will retain that transfer address.

SAVE.LOW

There is another form of the SAVE command resident in the UCS It is called SAVE.LOW and loads in a lower section of memory than the standard SAVE command. Its use is for saving programs in the Utility Command Space where SAVE.CMD is loaded. Those interested in creating their own utility commands should consult the 'Advanced Programmer's Guide' for further details.

STARTUP

STARTUP is not a utility command but is a feature of FLEX. It is often desirable to have the operating system do some special action or actions upon initialization of the system (during the bootstrap loading process). As an example, the user may always want to use BASIC immediately following the boot process. STARTUP will allow for this without the necessity of calling the BASIC interpreter each time.

DESCRIPTION

FLEX always checks the disk's directory immediately following the system initialization for a file called STARTUP.TXT. If none is found, the three plus sign prompt is output and the system is ready to accept user commands. If a STARTUP file is present, it is read and interpreted as a **single** command line and the appropriate actions are performed. As an example, suppose we wanted FLEX to execute BASIC each time the system was booted. First it is necssary to create the STARTUP file:

```
+++BUILD,STARTUP
  =BASIC
  =#
+++
```

The above procedure using the BUILD command will create the desired file. Note that the file consisted of one line (which is all FLEX reads from the STARTUP file anyway). This line will tell FLEX to load and execute BASIC. Now each time this disk is used to boot the operating system, BASIC will also be loaded and run. Note that this example assumes two things. First, the disk must contain DOS.SYS and must have been LINKed in order for the boot to work properly. Second, it is assumed that a file called BASIC.CMD actually exists on the disk.

Another example of the use of STARTUP is to set system environment parameters such as TTYSET parameters or the assigning of a system and working drive. If the STARTUP command consisted of the following line:

```
TTYSET,DP=16,WD=6Ø:ASN,W=1:ASN:CAT,Ø
```

each time the system was booted the following actions would occur. First, TTYSET would set the 'depth' to 16 and the 'width' to 6Ø. Next, assuming the 'end of line' character is the ':', the ASN command would assign the working drive to drive 1. Next ASN would display the assigned system and working drives on the terminal. Finally, a CATalog of the files on drive Ø would be displayed. For details of the actions of the individual commands, refer to their descriptions elsewhere in this manual.

As it stands, it looks as if the STARTUP feature is limited to the execution of a single command line. This is true but there is a way around the restriction, the EXEC command. If a longer list of operations is desired than will fit on one line, simply create a command file containing all of the commands desired. Then create the STARTUP file using the single line.

```
EXEC,(file name)
```

where (file name) would be replaced by the name assigned to the command file created. A little imagination and experience will show may uses for the STARTUP feature.

By directing STARTUP to a file that does not have a return to DOS command it is possible to lockout access to DOS. You can correct the problem by hitting the RESET button, setting the program counter addresses AØ48 and AØ49 to 71Ø3 and typing G for go. The STARTUP file may then be deleted and if desired, modified. Directing execution to 71Ø3, the DOS warm start address, bypasses the DOS STARTUP function.

TTYSET

The TTYSET utility command is provided so the user may control the characteristics of the terminal. With this command, the action of the terminal on input and the display format on output may be controlled.

DESCRIPTION

The general syntax of the TTYSET command is:

TTYSET{,(parameter list)}

where (parameter list) is a list of 2 letter parameter names, each followed by an equals sign ('='), and then by the value being assigned. Each parameter should be separated by a comma or a space. If no parameters are given, the values of all of the TTYSET parameters will be displayed on the terminal.

The default number base for numerical values is the base most appropriate to the parameter. In the descriptions that follow, 'hh' is used for parameters whose default base is hex; 'dd' is used for those whose default base is decimal. Values which should be expressed in hex are displayed in the TTYSET parameter listing preceded by a '$'. Some examples follow:

```
+++TTYSET
+++TTYSET,DP=16,WD=63
+++TTYSET,DX=F,BS=8,ES=3
```

The first example simply lists the current values of all TTYSET parameters on the terminal. The next line sets the depth 'DP' to 16 lines and the terminal width, 'WD' to 63 columns. The last example sets the computer duplex, 'DX' to Full, the backspace character to the value of hex 8, and the escape character to hex 3.

The following fully describes all of the TTYSET parameters available to the user. Their initial values are defined as well as any special characteristics they may possess.

BS=hh BackSpace character

This sets the 'backspace' character to the character having the ASCII hex value of hh. This character is initially a 'control H' (hex Ø8), but may be defined to any ASCII character. The action of the backspace character is to delete the last character typed from the terminal. If two backspace characters are typed, the last two characters will be deleted, etc. Setting BS=Ø will disable the backspace feature.

DL=hh DeLete character

This sets the 'delete current line' character to the hex value hh. This character is initially a 'control X' (hex 16). The action of the delete character is to 'erase' the current input line before it is accepted into the computer for execution. Setting DL=Ø will disable the line delete feature.

EL=hh End of Line character

This character is the one used by FLEX to separate multiple commands on one input line. It is initially set to a colon (':'), a hex value of 3A. Setting this character to Ø will disable the multiple command per line capability of FLEX. The parameter 'EL=hh' will set the end of line character to the character having the ASCII hex value of hh. This character must be set to a printable character (control characters not allowed).

DP=dd DePth count

This parameter specifies that a page consists of dd (decimal) physical lines of output. A page may be considered to be the number of lines between the fold if using fan folded paper on a hard copy terminal, or a page may be defined to be the number of lines which can be displayed at any one time on a CRT type terminal. Setting DP=Ø will disable the paging (this is the initial value). See EJ and PS below for more details of depth.

WD=dd WiDth

The WD parameter specifies the (decimal) number of characters to be displayed on a physical line at the terminal (the number of columns). Lines of text longer than the value of width will be 'folded' at every multiple of WD characters. For example, if WD is 5Ø and a line of 125 characters is to be displayed, the first 5Ø characters are displayed on a physical line at the terminal, the next 5Ø characters are displayed on the next physical line, and the last 25 characters are displayed on the firs physical line. If WD is set to Ø, the width feature will be disabled, and any number of characters will be permitted on a physical line.

NL=dd NuLl count

This parameter sets the (decimal) number of non-printing (Null) 'pad' characters to be sent to the terminal at the end of each line. These pad characters are used so the terminal carriage has enough time to return to the left margin before the next printable characters are sent. The intial value is 4. Users using CRT type terminals may want to set NL=Ø since no pad characters are usually required on this type of terminal.

TB=hh TaB character

The tab character is not used by FLEX but some of the utilities may require one (such as the Text Editing System). This parameter will set the tab character to the character having the ASCII hex value hh. This character should be a printable character.

DX=F or DX=H set DupleX

This parameter sets the computer transmission mode to either Full or Half duplex. A terminal may also operate in either mode of duplex. Some terminals have a switch enabling the operator to determine in which mode the terminal is to operate. When a terminal operates in full duplex mode, depressing a character key transmits the character to the computer, but the character is not printed at the terminal. The terminal expects the computer to 'echo' (transmit back to the terminal) the character it receives. This echoed character is then printed. When a terminal operates in half duplex mode, depressing a character key transmits the character to the computer and prints the character at the terminal -'echo' is not expected.

The computer may also operate in either full or half duplex mode. In full duplex mode, each character received by the computer is 'echoed' (transmitted back) to the terminal. In half duplex mode, the characters received are not echoed. (*note: Computers with a Control Interface, MP-C, installed instead of the Serial interface, MP-S, should have the duplex set to HALF. The computer will still echo characters and act as if it were in the full duplex mode. This is a characteristic of the hardware.)

The table below describes the effect of each of the four possible terminal and computer mode combinations:

TERMINAL	COMPUTER	EFFECT
full	full	the terminal prints normally since the computer is echoing all received characters
full	half	no input characters are printed
half	full	the terminal will either double-print or garbage-print input characters.
half	half	the terminal prints normally since the characters are printed as they are typed

It is best to operate both the terminal and the computer in full duplex mode since the character printed is a copy of what the computer actually received. This provides a double-check of the input characters.

EJ=dd EJectcount

This parameter is used to specify the (decimal) number of 'eject lines' to be sent to the terminal at the bottom of each page. If Pause is 'on', the 'eject sequence' is sent to the terminal after the pause is terminated. If the value dd is zero (which it is by default), no 'eject lines' are issued. An

eject line is simply a blank line (line feed) sent to the terminal. This feature is especially useful for terminals or printers with fan fold paper so as to skip over the fold (see Depth). It may also be useful for certain CRT terminals to be able to erase the previous screen contents at the end of each page.

PS=Y or PS=N PauSe control

This parameter enables (PS=Y) or disables (PS=N) the end-of-page pause feature. If Pause is on and depth is set to some nonzero value, the output display is automatically suspended at the end of each page. The output may be restarted by typing the 'escape' character (see ES description). If pause is disabled, there will be no end-of-page pausing. This feature is useful for those using high-speed CRT terminals to suspend output long enough to read the page of text.

ES=hh EScape character

The character whose ASCII hex value is hh is defined to be the 'escape character'. Its intial value is $1B, the ASCII ESC character. The escape character is used to stop output from being displayed, and once it is stoped, restart it again It is also used to restart output after Pause has stopped it. As an example, suppose you are LISTing a long text file on the terminal and you wish to temporarily halt the output. Typing the 'escape character' will do this (this feature is not supported on computers using a Control Interface (MP-C) for terminal communications). At this time (output halted), typing another 'escape character' will resume output, while typing the RETURN key will cause control to return to FLEX and the three plus sign prompt will be output to the terminal. It should be noted that line output termination always happens at the end of a line.

VERIFY

The VERIFY command is used to set the File Management System's write verify mode. If VERIFY is on, every sector which is written to the disk is read back from the disk for verification (to make sure there are no errors in any sectors). With VERIFY off, no verification is performed.

DESCRIPTION

The general syntax of the VERIFY command is:

```
VERIFY{,ON}
  or
VERIFY{,OFF}
```

where ON or OFF sets the VERIFY mode accordingly. If VERIFY is typed without any parameters, the current status of VERIFY will be displayed on the terminal. Example:

```
+++VERIFY,ON
+++VERIFY
```

The first example sets the VERIFY mode to ON. The second line would display the current status (ON or OFF) of the VERIFY mode. VERIFY causes slower write times, but it is recommended that it be left on for your protection.

VERSION

The VERSION utility is used to display the version number of a utility command. If problems or updates ever occur in any of the utilities, they may be replaced with updated versions. The VERSION command will allow you to determine which version of a particular utility you have.

DESCRIPTION

The general syntax of the VERSION command is:

VERSION,(file spec)

where (file spec) is the name of the utility you wish to check. The default extension is CMD and the drive defaults to the working drive. As an example:

+++VERSION,Ø.CAT

would display the version number of the CAT command (from drive Ø) on the terminal.

GENERAL SYSTEM INFORMATION

1. DISK CAPACITY

Each mini diskette when used with the mini version of FLEX is capable of holding 612 sectors. Each sector can contain a maximum of 124 characters (4 bytes in each sector are used by the system). The total capacity of the diskette is then 75,888 characters or bytes of information. The other limitation on a diskette is that the directory will support a maximum of 75 files. Trying to put more files than that on a diskette will cause an error message to be issued. In most cases, this is not a restriction since the actual disk space will be used up before the directory space. Also, the disk becomes cumbersome to use after about 6Ø files have been created.

II. WRITE PROTECT

It is possible to write protect a disk (make it impossible for the system to write on a disk) by placing a piece of opaque tape over the small rectangular cutout on the edge of the diskette. Any attempts to write files or delete files on a protected disk will cause an error message to be issued. It is good practice to write protect disks which have important files on them.

III. THE 'RESET' BUTTON

The RESET button on the front panel of your computer should NEVER BE PRESSED DURING A DISK OPERATION. There should never be a need to 'reset' the machine while in FLEX. If the machine is 'reset' and the system is writing data on the disk, it is possible that the entire disk will become damaged. Again, never press 'reset' while the disk is operating! Refer to the 'escape' character in TTYSET for ways of stopping FLEX.

IV. NOTES ON THE P COMMAND

The P command tries to load a file named PRINT.SYS from the same disk which P itself was retrieved. The PRINT.SYS file which was supplied with the system diskette contains the necessary routines to operate a SWTPC PR4Ø printer connected through a parallel interface on PORT 7 of the computer. If you wish to use a different printer configuration, consult the 'Advanced Programmer's Guide' for details on writing your own printer driver routines to replace the PRINT.SYS file.

V. ACCESSING DRIVES NOT CONTAINING A DISKETTE

If a disk access is requested of a drive not containing a diskette, the FLEX operating system will lock up until a diskette is placed in this drive and the door closed. The RESET button on the computer may also be used to exit this type of lock-up. After pressing RESET the FLEX operating system should be re-entered by jumping to hex location 71Ø3.

VI . SYSTEM ERROR NUMBERS

All expected errors will be displayed on the terminal as a plain English type error message. Less frequent error types will be reported as an error number in the following manner:

DISK ERROR #xx

where 'xx' is a decimal error number. The table below is a list of these numbers and what error they represent.

ERROR#	MEANING
1	ILLEGAL FMS FUNCTION CODE
2	FILE CURRENTLY BUSY
3	FILE EXISTS
4	NO SUCH FILE
5	DIRECTORY ERROR
6	TOO MANY FILES (DIRECTORY FULL)
7	DISK FULL
8	END OF FILE REACHED
9	READ ERROR (CRC)
10	WRITE ERROR (CRC)
11	DISK IS WRITE PROTECTED
12	DELETE ERROR
13	ILLEGAL FILE CONTROL BLOCK
14	ILLEGAL DISK ADDRESS
15	DRIVE NUMBER ERROR
16	DRIVE NOT READY
17	FILE ACCESS DENIED
18	FILE STATUS ERROR
19	DATA INDEX RANGE ERROR
20	FMS INACTIVE
21	ILLEGAL FILE NAME
22	FILE CLOSE ERROR

For more details concerning the meanings of these error messages, consult the 'Advanced Programmer's Guide'.

VII . SYSTEM MEMORY MAP

The following is a brief list of the RAM memory space required by the FLEX Operating System. All address are in hex.

ØØØØ - 2FFF	User RAM *Note: Some of this space is used by CAT, COPY, APPEND, BACKUP, and NEWDISK
3ØØØ - 6FFF	UserRAM
7ØØØ - 75FF	Disk Operating System
71ØØ	FLEX cold start entry address
71Ø3	FLEX warm start entry address
76ØØ - 773F	Utility command space
774Ø - 777F	System FCB
78ØØ - 7EFF	File Management System
7FØØ - 7FFF	Disk driver routines
AØ4A - AØ7F	System stack

For a more detailed memory map, consult the 'Advanced Programmer's Guide'.

VIII. FLEX OPERATING SYSTEM INPUT/OUTPUT SUBROUTINES

In order for the TTYSET duplex setting function to operate properly, all user program character input/output subroutines should be vectored thru the FLEX operating system rather than the computer's monitor. Below is a list of FLEX's I/O jumps and a brief description of each. All given addresses are in hexadecimal.

71ØF

This subroutine is functionally equivalent to SWTBUG®'s or MIKBUG®'s character input routine E1AC. This routine will look for one character from the control terminal (I/0 #1) and store it in the A accumulator. Once called, the input routine will loop within itself until a character has been input. Anytime input is desired, the call JSR $71ØF should be used.

71ØF automatically sets the 8th bit to Ø and does not check for parity. When using the subroutine the processor's registers are affected as follows:

ACC A	loaded with the character input from the terminal
ACC B	not affected
IXR	not affected

7112

This subroutine is used to output one character from the computer to the control port (I/0 #1).

To use 7112 the character to be output should be placed in the A accumulator in its ASCII form. To output the letter A on the control terminal, the following program could be used:

```
LDA A   #$41
JSR     $7112
```

The processor's registers are affected as follows:

ACC A	changed internally
ACC B	not affected
IXR	not affected

This routine is functionally equivalent to E1D1 in SWTBUG® and MIKBUG® monitors.

7118

7118 is the entry point of the subroutine used to output a string of text on the control terminal. When address 7118 is called, a carriage return and line feed will automatically be generated and data output will begin at the location pointed to by the index register. Output will continue until a Ø4 is seen. The same rules for using the ESCAPE and RETURN keys for stopping output apply as described earlier.

The accumulator and register status after using 7118 are as follows:

ACC A	Changed during the operation
ACC B	UNCHANGED
IXR	Contains the memory location of the last character read from the string (usually the 04 unless stopped by the ESC key)

NOTE: The ability of using backspace and line delete characters is a function of your user program and not of the FLEX I/O routines described above. Correct operation of the TTYSET utility to set the desired duplex mode assumes that the monitor's "initialization" sequence is never invoked. (If a user program is ever entered by using the G or J function of the monitor, the duplex mode of the FLEX I/O routines may not be correct.) Any user program which is entered directly from FLEX will run correctly.

For additional information consult the 'Advanced Programmer's Manual'.

Below is a short bootstrap program which will load the FLEX operating system from the system diskette. This boot is not necessary for user's having a SWTBUG® monitor-SWTBUG® already contains this boot.

To bring up the FLEX operating system, enter the bootstrap program below instruction by instruction using the memory examine and change function of your monitor. As shown, the bootstrap loads from hex address Ø to 0149. After entering the bootstrap, set the computer's program counter A048 and A049 to Ø After a system diskette is installed in drive Ø, a G may be entered to execute the bootstrap.

If the system will not boot properly, re-position the system diskette in the drive and re-execute the bootstrap. The diskette to be booted must be initialized and must also contain the disk operating system software.

```
0100 B6 8018  START  LDA A COMREG  TURN MOTOR ON
0103 4F              CLR A
0104 B7 8014         STA A DRVREG
0107 CE 0000         LDX  #0
010A 08     OVR      INX
010B 09              DEX
010C 09              DEX
010D 26 FB           BNE  OVR
010F C6 0F           LDA B #$0F   RESTORE
0111 F7 8018         STA A COMREG
0114 8D 31           BSR  RETURN
0116 F6 8018 LOOP1   LDA B COMREG  LOOP UNTIL THRU
0119 C5 01           BIT B #1
011B 26 F9           BNE  LOOP1
011D 7F 801A         CLR  SECREG
0120 8D 25           BSR  RETURN
0122 C6 9C           LDA B #$9C   READ WITH LOAD
0124 F7 8018         STA B COMREG
0127 8D 1E           BSR  RETURN
0129 CE 2400         LDX  #$2400
012C C5 02   LOOP2   BIT B #2    DRQ?
012E 27 06           BEQ  LOOP3
0130 B6 801B         LDA A DATREG
0133 A7 00           STA A 0,X
0135 08              INX
0136 F6 8018 LOOP3   LDA B COMREG
0139 C5 01           BIT B #1    BUSY?
013B 26 EF           BNE  LOOP2
013D F6 8018         LDA B COMREG
0140 C4 0C           AND B #$0C   CHECK FOR CRC AND LDATA
0142 26 BC           BNE  START   RE-DO IF ERRORS
0144 7E 2400         JMP  $2400
0147 20 00   RETURN  BRA  RTN
0149 39      RTN     RTS
```

X. Requirements for the PRINT.SYS Printer Driver

FLEX, as supplied, includes a printer driver routine that will work with most parallel type printers, such as the SWTPC PR-4Ø. If desired the printer driver may be changed to accommodate other types of printers. Included is the source listing for the supplied driver. Custom drivers **must** adhere to the following rules for correct operation:

1.) The driver must be in a file called PRINT.SYS

2.) Hex location ØØ1Ø must contain the starting address of the port initialization routine.

3.) Hex location ØØ12 and location 71ØD must contain the address of the character output routine.

4.) When the printer character output routine is called by FLEX, the character to be output will be in the A accumulator. The output routine must not destroy the index register or the B accumulator.

5.) Both the initialization and output routine may reside anywhere in memory, but must not conflict with any utilities or programs which will use P.

6.) Both the initialization and the output routine must end with a return from subroutine RTS.

```
1                       NAM   PRINT
2                       OPT   PAG
3                *GENERATES PRINT.SYS FILE FOR P UTILITY
4                *VERSION 1

6 0001           VERSION EQU   1          SET UP VERSION NUMBER
7 801C           PORT    EQU   $801C      SET UP FOR PORT 7

9 0010                   ORG   $0010
10 0010 A0 4A            FDB   INIT       SET UP INITIALIZATION ADDRESS
11 0012 A0 16            FDB   OUTCHR     SET UP OUTPUT ADDRESS

13 710D                  ORG   $701D
14 710D A0 16            FDB   OUTCHR

16               *THIS ROUTINE OUTPUTS THE CONTENTS
17               *OF THE A ACCUMULATOR TO A PARALLEL PRINTER
18               *ASSUMES AN MP-L OR MP-LA INTERFACE ON PORT 7
19 A016                  ORG   $A016
20 A016 20 01    OUTCHR  BRA   OUT
21 A018 01               FCB   VERSION    SET VERSION BYTE
22 A019 FF A0 35 OUT     STX   SAVEX      SAVE IXR AND ACC B
23 A01C 37               PSH B
24 A01D CE 80 1C         LDX   *PORT      PORT *7
25 A020 A7 00            STA A 0,X        GIVE DATA TO PIA
26 A022 C6 36            LDA B *$36       GENERATE READY STROBE
27 A024 E7 01            STA B 1,X
28 A026 C6 3E            LDA B *$3E
29 A028 E7 01            STA B 1,X
30 A02A 6D 01    LOOP    TST   1,X        WAIT FOR ACCEPTED PULSE
31 A02C 2A FC            BPL   LOOP
32 A02E E6 00            LDA B 0,X
33 A030 FE A0 35         LDX   SAVEX      RECOVER IXR AND ACC B
34 A033 33               PUL B
35 A034 39               RTS
36 A035          SAVEX   RMB   2          TEMPORARY STORAGE
```

CONTINUED

CONTINUED FROM FRONT

```
    38 A04A                 ORG    $A04A
    39 A04A CE 80 1C  INIT  LDX    *PORT
    40 A04D C6 FF           LDA B  *$FF          ALL OUTPUTS
    41 A04F E7 00           STA B  0,X
    42 A051 C6 3F           LDA B  *$3E          SET UP READY STROBE
    43 A053 E7 01           STA B  1,X
    44 A055 39              RTS
    45                      END

NO ERROR(S) DETECTED
```

COMMAND SUMMARY

NOTES: