commodore

COMMODORE BUSINESS MACHINES LIMITED
3370 PHARMACY AVENUE, AGINCOURT, ONTARIO M1W 2K4
TELEPHONE (416) 499-4292 – CABLE ADDRESS: COMTYPE
TELEX NUMBER 06-525400

Bulletin 4

TABLE OF CONTENTS

1. ADDITIONAL BASIC COMMANDS

By this time, the user is probably familiar with the use of the commands INPUT and PRINT. INPUT permits the entry of data from the input keyboard and PRINT permits the output or display of data. These commands are common to all forms of BASIC.

To add flexibility to the PET computer system, several commands have been added to classical BASIC in the PET and future Commodore products will take advantage of the resulting extra capability. In general, enhanced flexibility of data interchange between the PET and peripheral devices is possible, thanks to the use of these extra commands.

To communicate with any device, a combination of the additional commands is used:

(1)	OPEN/CLOSE	Open or close logical file.
(2)	PRINT#	Write data from PET to I/O device.
(3)	CMD	Same as PRINT# but leaves IEEE device on bus after execution of command.
(4)	INPUT#	Read data from I/O device to PET.
(5)	GET#	PET accepts one character from I/O device.

2. INPUT/OUTPUT COMMAND PARAMETERS

In order to use the additional commands referred to in Section 1, four parameters must be taken into consideration:

(1) Logical file number (L)
(2) Device number (D)
(3) Secondary address (SA)
(4) File-name (FN)

These parameters can appear, for example, when using the OPEN# command in the form of the statement:

```
OPEN#L,D,SA,FN
```

These parameters are defined and explained in sections 2.1 through 2.4. The default values for these parameters are listed in Tables 12-1 and 12-2 of the Appendix.

2.1 LOGICAL FILES

Files are used to store and retrieve data, as for example in the case of a magnetic tape or disc file. A convenient extension of this idea is to regard any device which can

receive and/or generate data as a logical file. To the PET operating system, data might just as well have come from, or be going to, a storage system such as magnetic tape.

For example, imagine that an external digital voltmeter is set up so that it can transmit voltage readings upon request to the PET via the IEEE data bus. Sometime during the "voltmeter program" the programmer will have to open a file and assign a logical file number to the voltmeter. Once this has been done, the PET can use a 'read' command (INPUT#) that uses the logical file number to refer to the voltmeter. When no further data is required from the voltmeter, the logical file can be closed.

More generally, the advantages offered by the use of logical files are:

(a) Every device number-secondary address combination can be associated with its own unique logical file number within a program.

(b) Multiple files within a single device can be referred to by means of distinct logical file numbers. This approach is to be used in the newly developed disc storage system for the PET.

(c) Once a logical file number has been defined in an OPEN statement within a program, only this single number need be used in the following input/output statements. This eliminates the need for further restatement of device number, secondary address (where used) and file name (where used).

Although it is permissible to identify and use many logical files in a given program, the PET operating system has to keep track of the files that are currently in use in the program. The greatest number of files that can be controlled by the PET at one time is ten. Note that in the present version of the operating system, exceeding ten will result in loss of PET operation; this can be restored by switching the computer off and on. A logical file number can be any integer in the range 1 through 255.

2.2 DEVICE NUMBER

All devices which the PET communicates with are assigned device numbers. The first four of these are reserved for the following peripherals:

	DEVICE NUMBER	DEVICE
	0	Keyboard
Default -	1	Cassette 1 panel mounted.
	2	Cassette 2 add-on.
	3	Video screen.

All other devices are automatically assumed by the PET to be IEEE devices, and control is transferred to the device which will have been allocated a number within the range 4 through 30. Except in special cases, a specific number would be allocated to each IEEE device to allow the PET and a particular device to communicate using the parallel IEEE-488 bus.

On many IEEE devices, the allocation of the device number is made by means of a switch, or in the case of less expensive products, by the connection of jumpers.

2.3 SECONDARY ADDRESSES

The concept of a secondary address may be new to those people who have never worked with the IEEE bus. The use of a secondary address permits an intelligent peripheral to function in any one of the number of modes. For example, in the PET 2020 printer, there are six secondary addresses:

	SECONDARY ADDRESS	OPERATION
Default -	0	Normal printing.
	1	Printing under format statement control.
	2	Transfer data from PET to format statement.
	3	Set variable lines per page.
	4	Use expanded diagnostic messages.
	5	Byte data for programmable character.

In short, by changing the secondary address used to communicate with a given physical device, its operating characteristics can be totally changed, if so desired. Many of the IEEE devices have their own particular secondary address conventions which must be followed. Specific data on these conventions can be obtained by consulting the manual for that particular device.

The PET tape units have a special set of secondary address rules:

SECONDARY ADDRESS	OPERATION
Default - 0	Tape is being opened for 'read'.
1	Tape is being opened for 'write'.
2	Tape is being opened for 'write' with an 'end of tape' header being forced when the file is closed.

The secondary address can have values over the range 0 through 31.

2.4 FILE NAMES

In random storage devices where there is more than one file to be accessed, the use of names to identify files is mandatory. In the case of tapes, a file name is desirable, even if there is only one file on the tape, since it facilitates the identification of tapes.

For the two cassette tape units of the PET, a file name may be any combination of up to 128 characters.

When a file name is searched for, it is matched on an ascending character basis. Assume that an eight character file name COUNTING was specified when writing, if desired this could be searched for with an abbreviated name such as COU. The first file header that is found with these three consecutive characters will then be opened and positioned on. In principle, this could include unwanted file names such as COUNT or COUNTRY, as well as the wanted COUNTING. It is, therefore, advisable to specify the complete file name in order to avoid errors.

For other devices which use named files, the individual description of the device should be consulted in order to ascertain the specific requirements for file name usage.

3. TAPE CASSETTE FILE OPERATION

The PET devotes special attention to the two tape cassette units that can be attached to it. The tape units are specially modified so that the PET has control over the motor movement of the cassette. If can also sense when the play, rewind, or fast forward buttons have been pushed; this is done by means of a single switch mounted in the tape unit. Note that the same switch is used to sense all three buttons, and if any of the three is pushed, the PET will think

that you pushed the 'play' key and will respond accordingly.

Because of the type of mechanism used in the tape unit, the user must rewind, fast forward, stop, load and eject tapes. He must also put the unit into the write mode by pushing the record button either simultaneously with, or before the 'play'button is pressed.

The PET has total control over the movement of the tape once the appropriate buttons have been pushed to engage the motor.

Messages displayed throughout the program will tell the user when it is necessary for him to initiate the function of play or record. Logic dictates the times when the user should rewind and fast forward.

The two tape units of the PET are handled independently, and the various control lines permit the reading of data from cassette #1, the reading of data from cassette #2, motor control of cassette #1, motor control for cassette #2 and a common write line.

3.1 RECORDING METHOD

The data structure embodied in tape files will ensure the maximum memory utilization and maximum reliability of recording.

To accomplish this, the PET records data at two audio frequencies in two consecutive blocks of data. All of the data is totally repeated, and by means of error checking techniques incorporated in the PET software, it is possible for most audio dropouts to be corrected by the operating system utilizing the redundant data stored in the second data block.

In order to correct for (a), the fact that tape units record at different speeds, and (b), the normal drag characteristics of tapes, a speed correlation technique is used during reading. To correct for the individual start/stop characteristics on the tape and synchronize correctly between each block on tape, a single tone is written between blocks. This signal is used to synchronize both position and speed of the tape. Varying lengths of tone are used at the beginning and between the data blocks of the tape. By writing about ten seconds of the tone on each opening of a file, the PET automatically corrects for normal leader. Individual tape blocks are separated by shorter tone durations.

3.2 FILE HEADERS

An important assumption underlying the tape system design was that the user would often want to record more than one file of data on tape. In order to facilitate this and to allow for proper label checking, the first physical data recorded on tape for any operation is a file header. This file header looks exactly the same as the normal data block, except that the first character of every block on tape contains an identification character which enables the operating system to differentiate between program blocks, data blocks, file headers and end of tape headers.

The PET allows for up to 128 characters of a file name to be stored in the file header. This is the name which is searched for and matched on in the various OPEN/CLOSE options.

3.3 TAPE BUFFERS

Another basic premise in the design of the tape operating system was that the user would want to write tape independently of what is occurring on tape at a given moment. This is accomplished in the operating system by permanently assigning a block of memory as a data buffer for each cassette tape. A 192 character buffer is located at decimal address 634 for cassette #1, followed by a 192 character buffer at decimal address 826 for cassette #2. The tape file header is written into the buffer first and then written on tape.

Data files are accumulated in the tape buffer until 192 characters are exceeded, then the contents are either written on tape for write, or if the program is reading tape, the next block of data is read into the buffer. Tape file headers and all data blocks are, therefore, 192 characters long.

Tape buffers are not used in the case of program files, since these are written onto the tape directly from the memory in which the program resides. In order to accommodate the variable memory location, the file header for a program file contains the beginning and ending address for the program. The full program is written onto tape in the usual form of two consecutive redundant blocks.

3.4 MULTIPLE FILES

In order to have multiple files on tape, the user needs the ability to add files to a tape and also know when the tape is at its end. It is, therefore, important that the operating system give an 'end of file' and end of tape indication.

In the case of data files, and 'end of file' marker is appended after the last data character. This is available to the user in a status word on reading; the 'end of file' marker is automatically inserted when a write file is closed.

In the case of program files, because all the data is always contained in a single block, the end of the block signifies the end of the program.

To signify that the end of tape has been reached, a special separate file header is written. When this is encountered during a search for files, the PET automatically stops the tape and indicates 'file not found' to the user. A typical multiple file tape could contain first a data file, then a program file, followed by an 'end of tape' header as illustrated in the example of Figure 3-1.

FIGURE TAPE SYSTEM

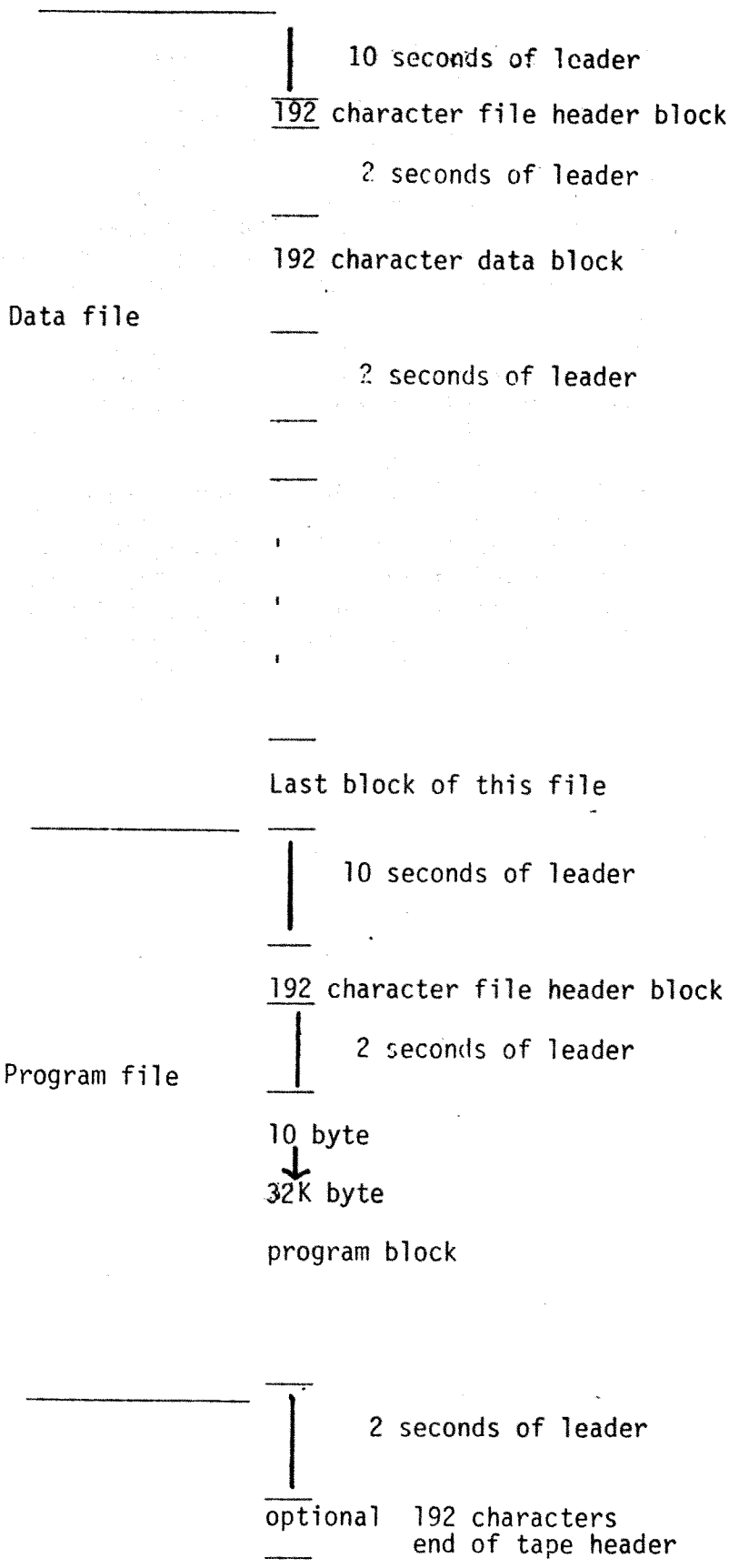

Figure 3-1. An exa of multiple file structure.

4. LOGICAL FILE INPUT/OUTPUT OPERATIONS: GENERAL ORIENTATION

These operations can be subdivided into three steps:

(1) Open the file - tell the PET everything it needs to know about the file.
(2) Read data from, or write data to the logical files.
(3) Close the file - allow the PET to clear up the device and terminate the active file.

These steps are discussed in detail in Sections 5. through 9.

5. OPENING FILES

In order to tell BASIC about the file you want to operate on, it is first necessary to open the file. This is done by the following statement:

OPEN Logical file, device, secondary address, file name

More specifically, the statement consists of the command OPEN followed by the logical file number, then the device number to which the file is assigned, then the secondary address data (if any) communicated during the interaction of BASIC with the file, and last the name of the physical file (if any).

This statement, or expression, is interpreted by BASIC, and could, therefore, use computed logical file numbers, device numbers or secondary address data. This capability is extremely useful when handling multiple file devices such as discs.

The keyword 'OPEN' and the logical file numbers are essential in order to open a file; that is address a device in preparation for a 'read' (INPUT#) or a 'write' (PRINT#).

The device number is optional; if not entered, the default value '1' will be used (see Section 2.3 and Appendix).

A file name is optional, though preferred, for the tape units; a name would be essential for a disc storage unit, however.

5.1 EXAMPLES OF OPEN STATEMENTS

The statement OPEN 1,2,1 is interpreted by the operating system as saying:

PARAMETER
(L) Logical file #1 has been opened.
(D) Logical file #1 has been assigned to tape unit #2.
(SA) Tape unit #2 has been instructed to write on tape.
(FN) A file name has not been assigned to the tape record.

Similarly OPEN 3 is interpreted as saying:

PARAMETER
(L) Logical file #3 has been opened.
(D) Logical file #3 has been assigned to tape unit #1 (default '1').
(SA) Tape unit #1 has been instructed to read from tape (default '0').
(FN) No file name referred to.

If the PET 2020 printer is assigned '4' as a device number, then OPEN 12,4,1 is interpreted as:

PARAMETER

(L) Logical file #12 has been opened.

(D) Logical file #12 has been assigned to device #4.

(SA) Printer has been instructed to print under format statement control (see Section 2.3).

(FN) File name not applicable.

NOTE: The current version of PET has a problem with OPEN for tape files. The opening of the tape file is automatic, but the tape header may not always be written at the beginning of the tape buffer; this implies that the operating system does not always correctly initialize the buffer pointer. For consistent and reliable operation of the tape file header, the following statements should be used:

```
(1) For tape #1:    POKE 243,122
                    POKE 244,2
(2) For tape #2:    POKE 243,58
                    POKE 244,3
```

These should be written prior to each OPEN for write.

This problem will be resolved in due course as a set of modified ROMs will shortly be available. However, the two POKEs will not cause any PET malfunction, if the new ROMs are installed.

5.2 LOAD

A special case of the OPEN command is the LOAD of a named file; a LOAD is done with the following statement:

```
LOAD name, device number
```

The operating system automatically generates an OPEN using the appropriate secondary addresses for 'load'. This open causes the loading device to search for a program name. After the program is found, it is automatically read from the device and loaded into memory starting at an address specified in the file header. Any reading errors on the first pass through that program are automatically fixed on the second pass, if possible.

At the end of the load cycle, a checksum of the total program is made. If a checksum error, or if an unrecoverable read error occurred, the operating system automatically prints '?LOAD ERROR' and stops the load program.

If the program load was from direct mode, the clear function is performed at the end of the load, thereby, initializing all variables.

If the load is called from a program, then the PET treats this load as an overlay. The new program is loaded into the space used by the previous program but the values of all of the variable are maintained from the previous program. This allows for one program to call another and pass parameters to the called programs.

The only restriction on this is that all called programs must fit in the same, or less space as the first program.

Because BASIC totally replaces the current program, it is not directly possible to have a single main program and several subroutine overlays, however, by including the main program with each overlay, the same effect is obtained with some loss of speed.

The combination of the use of named files and overlays allows the writing of very large structured programs of significant complexity.

5.3 VERIFY

This instruction is a special case of LOAD. It should be used after every program SAVE.

The command causes BASIC to go through all the steps of a program LOAD, with the exception that the data does not get loaded into memory, but, instead, gets compared with memory. If either first or second pass errors occur, the PET will type out '?VERIFY ERROR' which means that the program should be saved again before it is lost. On verify, the status word has the following meanings (see Section 10.1 for explanation of meaning):

CODE	MEANING
4	short block
8	long block
16	any mismatches
32	checksum ERROR on tape

5.4 SAVE

'SAVE' also performs an automatic open and close. The SAVE is specified by the statement:

SAVE name, device number

If the physical device is one of the two tape units, the operating system automatically initiates a tape header and opens a tape file with the appropriate name. The file header is written with the beginning and ending address.

If the device is an IEEE-488 device, a special open message is sent indicating that the PET is sending a program file.

The program is then written directly from its memory locations to the tape or the IEEE-488 bus.

If the SAVE is on tape, a checksum is computed and also saved and then the whole program is, again, written to give the redundant recording. At the end of the program, the tape is automatically stopped and positioned for the next data.

IEEE-488 SPECIAL FEATURES

In the tape, the program beginning and ending address are stored in and retrieved from the tape file header.

In order to more efficiently use the IEEE-488 data, the starting address of the program is sent as the first two bytes of data on a SAVE and retrieved from those positions on a LOAD.

5.5 IEEE-488 OPEN CONSIDERATIONS

If the OPEN command selects a device which has a value of 4 or more, the operating system assumes that the device is an IEEE-488 device.

If the OPEN does not specify a file name, then nothing is communicated on the IEEE-488 bus. However, if a file name is specified, the operating system sends a listen attention sequence to the device number specified in the OPEN along with a secondary address which is the OR of hexadecimal "FØ" and the secondary address specified in the OPEN statement.

Commodore supplied peripherals, such as the floppy disc storage system, will use this secondary address and also the file name which is then transmitted to the listening device, in order to transfer data later to the open file.

6. TAPE FILE OPERATION MODES

Tape files can be opened for two distinct purposes:

(a) In order to write from the PET onto tape.
(b) In order to read from tape to the PET.

6.1 OPEN FOR WRITE ON TAPE FROM PET

The flow diagram of Figure 6-1 outlines the PET-user interaction and PET function when opening a file for write on tape. The initial block shows that there are two ways of opening the file:

(1) OPEN for write - data tape.
(2) SAVE - write a program tape.

Note that if the tape file is opened directly, that is from the keyboard, then the message 'writing name' is displayed. If the file is opened under program control, and the 'play and record' buttons are depressed previously, then no message appears on the screen; in this manner, any display material placed there by the current program is not disturbed.

6.2 OPEN FOR READ FROM TAPE TO PET

The flow diagram of figure 6-2 outlines the PET-user interaction and PET function when opening a file for reading on tape. The initial block shows that there are two ways of opening the file:

(1) OPEN for read data tape.
(2) LOAD program into memory.

Note that if the file is opened directly, that is from the keyboard, then the messages PRESS PLAY, SEARCHING FOR NAME and FOUND NAME are displayed. If LOAD was used, then the BASIC variables of the loaded program are initialized.

If the file is opened under program control and provided that the PLAY button had been pressed previously, no messages appear on the video screen in order not to disturb material displayed by the current program. Initialization of the BASIC variables does not occur.

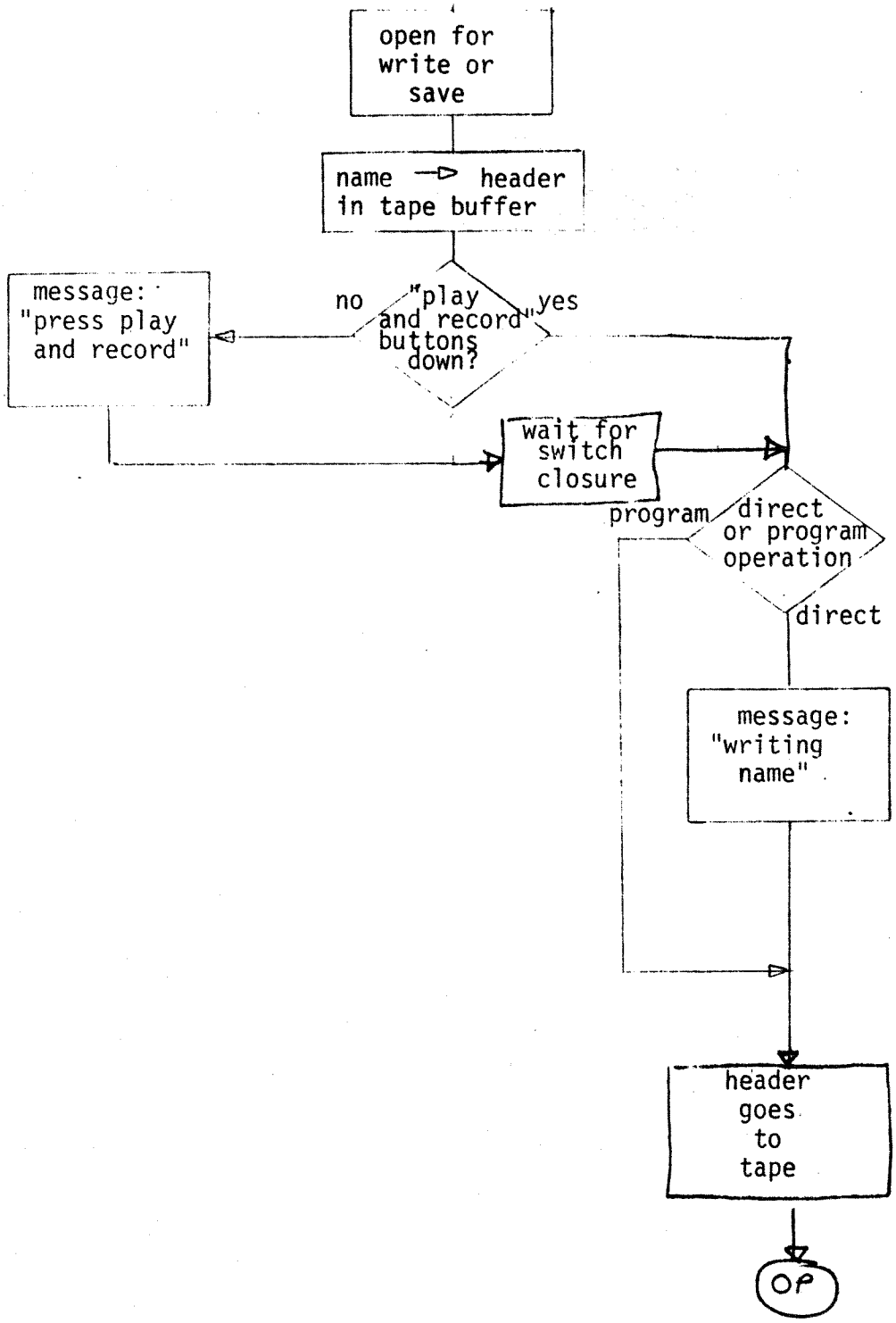

Figure 6-1. Open for write from PET. (PRINT#, CMD) or SAVE (op) = operating system takes over.

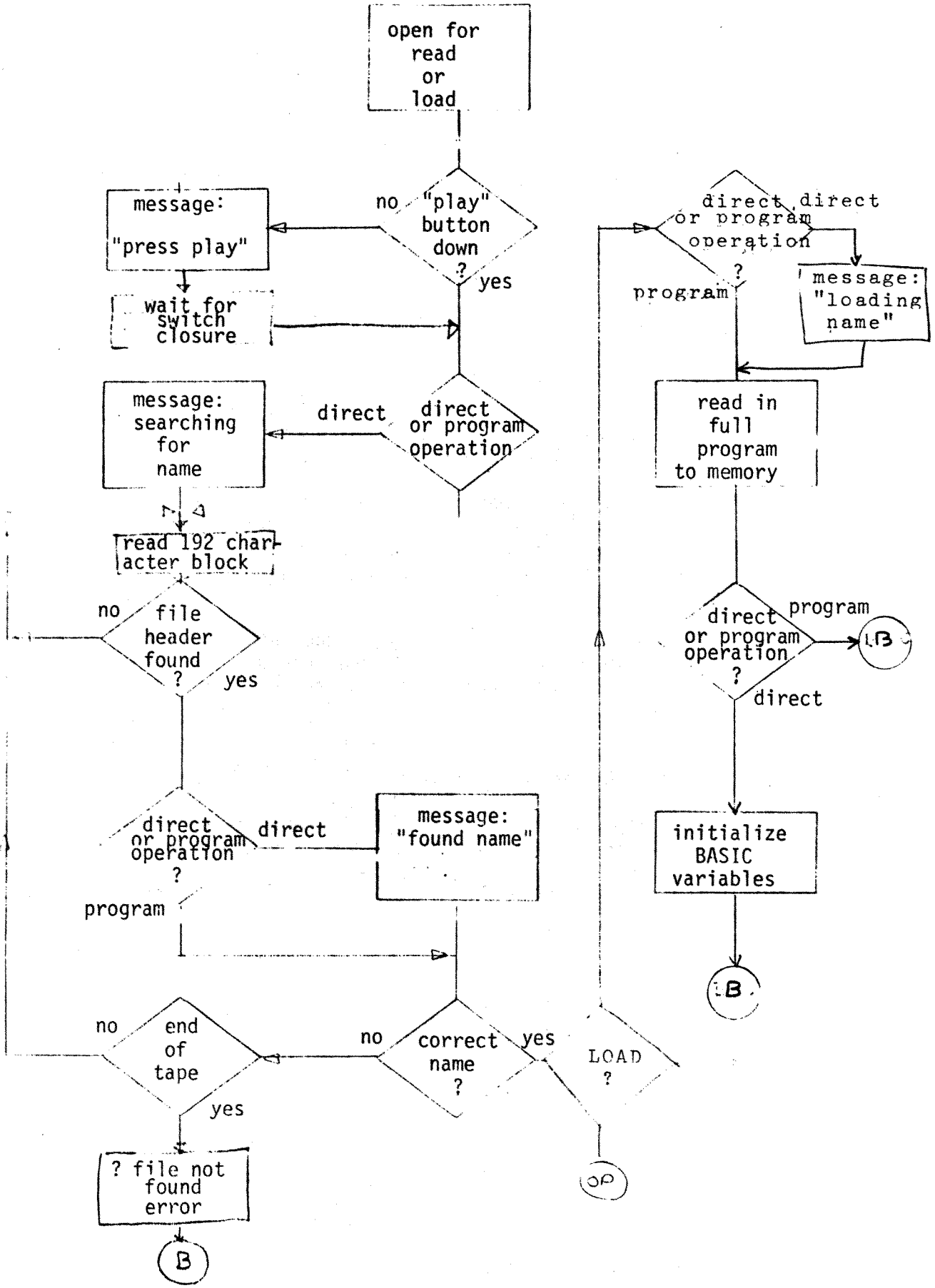

Figure 6-2 OPEN for read to PET (INPUT#) or load.
OP Operating system takes over.
B BASIC takes over

7. DATA INPUT

The use of the word 'input' in this context implies input of data to the PET from any device.

7.1 INPUT# - STRING AND VARIABLE INPUT

INPUT# is the command used to initiate data transfer from I/O devices to the operating system. The statement format is:

INPUT# logical file number, A,A$,B,B$, etc

Where A,A$,B and B$ are numerical and string variables to be inputted (read) from the selected logical file to the operating system one character at a time.

Because the rules for the BASIC interpreter apply to these input statements, all carriage returns, commas, terminate fields, nulls, preceeding blanks (except in strings), and other control characters are automatically deleted.

It is not always possible to mix both numeric and alphabetic data on an I/O device. If a numeric field is specified, only numeric data in the standard from expected by BASIC is accepted, otherwise a '?BAD DATA ERROR' message is displayed.

If there is any ambiguity about the data coming in, the user should input only to strings and then use the various string manipulation commands to process the data into the appropriate variables.

7.1.1 EXAMPLE OF INPUT# STATEMENT

If X represents a series of 50 numbers stored on a tape file named 'vector' and we assume that the 'play' button has just been depressed on tape unit #1. Then the following program will read the 50 numbers one at a time and display them on the video screen.

```
10  OPEN 1,1,0, 'VECTOR'      Open logical file #1.
                              Assign file to cassette 1.
                              Open tape for 'read'.
                              Look for physical file
                              named 'VECTOR'.

20  FOR K=1 to 50             Read 50 numbers one at a
                              time from cassette 1.

30  INPUT#1,X

40  PRINT X                   Display numbers on video
                              screen.

50  NEXT K

60  CLOSE 1                   When fifty numbers have
                              been read, close logical
                              file #1.
```

7.2 CHARACTER TRANSFERS - GET#

Not all devices transfer data in a form which is acceptable directly to BASIC. There is a series of binary data and combinations which BASIC ignores and although many IEEE devices do correctly respond with data formats which are acceptable to BASIC, not all do.

In addition, in some cases, it is desirable for the programmer to have immediate access to characters as they are transferred to the system. GET# fetches from the IEEE-488, or a tape device, a single character at a time, putting a character in a field specified following the GET#. The form is:

```
GET#  logical device,field
```

7.3 TAPE INPUT

When reading from the tape file, the data comes to the user I/O independent. Each time BASIC starts on INPUT# or GET# from a logical device which was opened for read on tape 1 or 2, a special subroutine is called, which initiates tape INPUT.

As each character is requested from BASIC, it is fetched from the appropriate tape buffer. When the buffer is empty, the tape INPUT routine suspends the user program and reads the next data block from tape into the buffer and then transfers the next character to BASIC. If a read error occurs, it is noted in the status word.

When the end of file mark is encountered in the buffer, the end of file position of the status word is set on and carriage returns are forced automatically out until the command is finished.

At the end of a command, BASIC calls another routine which reinitializes the input to the keyboard and tells the end of file operation that a command is complete.

7.4 IEEE-488 DEVICE INPUT SEQUENCES

The sequence that all INPUT# or GET#commands goes through is the same. When the command is first encountered, the IEEE-488 input initiation routine is called, which sends a talk attention sequence to the device and secondary address which was specified for that logical file in the open sequence. At the end of the attention sequence, the PET establishes itself in a listener mode and attempts to wait for a DAV signal indicating a single character has been received. If the DAV is received within 65 milliseconds, that character is handed to BASIC and/or to the other program calling the IEEE-488 routine. Each time the IEEE-488 routine is called, it will go through the same sequence of getting a single character while waiting for a time out to occur. If the bus does not respond in 65 milliseconds, then the IEEE-488 routine will automatically terminate the sequence; giving a read error in the status word to indicate that it has terminated the sequence.

If during the course of reading the character, the IEEE-488 routine senses an EOI line, it will indicate the end of information in the status word and will continue to return carriage returns, until the command it has been currently operating under has been terminated. At the end of the command, BASIC calls a termination subroutine which reinitializes the device to the keyboard and sends an untalk to the IEEE-488 bus, thereby, freeing the bus for the next command.

7.5 INPUT BUFFER LIMITATIONS

Although data is transferred from the operating system one character at a time, in order to edit, BASIC accumulates these characters into an 80 column INPUT buffer. This buffer must be terminated by a carriage return.

On current PETs, should more than 80 characters be read, the operating system will malfunction, as the operating system variables are overwritten. The PET can be made to function again by switching the line supply off and on. .

Although this problem will be resolved in future versions, the 80 column limitation will still apply. This constraint must be kept in mind when using tape and disc file systems.

This means that carriage returns must be written on tapes, discs, or other I/O devices in such a way that not more than 80 characters per field are written.

If an I/O device sends more than 80 characters, the GET command can be used to build your own string without running into the buffer limitation.

8. DATA OUTPUT

The use of the words 'print' and 'write' refers to data output from the PET to any device.

8.1 PRINT#

The command PRINT# must be followed by the logical file number, and then a comma to separate the data that would follow PRINT:

```
PRINT# logical file number, data
```

Data is transferred a single character at a time to the physical device correlated with the logical file specified in the relevant OPEN statement. Many of the file delimiters such as commas are automatically deleted by BASIC; although this does not greatly effect the printing, it should be remembered that when reading back from tape or another I/O device that file delimiters must be forced. This forcing can be done by inserting a CHR$(44) or "," between fields or by only printing single fields in each PRINT# statement which will force carriage returns between fields. Example:

instead of writing

```
PRINT# LF, A;B$;C$
```

which will be sent as:

```
AB$C
```

with no delimiters:

```
PRINT# LF, A; CHR$(44);B$;CHR$(44);C$
```

or:

```
PRINT# LF,A",";B$;",";C$
```

which will output:

```
A  ,B$,C$ (CR)
```

or:

```
PRINT#LF, A
PRINT#LF, B$
PRINT#LF, C$
```

which will output:

```
A (CR) B$ (CR) C$ (CR)
```

Because BASIC always formats outputs to any devices as though it were outputting to the screen, PRINT#LF, A,B has several skip characters between the values of A and B, while A ; B does not have any extra skips.

An exception to this rule is the tape where the first skip on output is suppressed.

NOTE: Although both the INPUT# and PRINT# commands operate in virtually the same way as their equivalent INPUT and PRINT statements do in BASIC, the abbreviated command '?' which can be used in place of PRINT, does not apply to PRINT#. ?# and PRINT# are recognized and reduced to two different token characters when processed by BASIC. ?# will look like PRINT# when listed but gives '?SYNTAX ERROR' when an attempt is made to execute it.

8.1.1 EXAMPLES OF PRINT# STATEMENT

This program will print the series of numbers 1,2,3,. . . 50, one at a time on the PET 2020 printer.

```
10  OPEN 5,4,0        Open logical file #5.
                      Assign logical file #5 to device
                      #4 (2020 printer) in normal print
                      mode corresponding to secondary
                      address '0'.

20  FOR K=1 to 50     Print the series of fifty numbers
                      on printer.

30  PRINT#5,K

40  NEXT K

50  CLOSE 5           Close logical file #5.
```

To write the above series of numbers on a cassette in tape unit #2, only the OPEN line would have to be modified, if the same logical file number were chosen:

```
10  OPEN 5,2,1        Open logical file #5 "new line".
                      Assign logical #5 to device #2
                      (tape unit #2) with a write
                      without "end of tape" designation
                      corresponding to secondary
                      address '1'.

20  FOR K=1 to 50     Record the series of fifty
                      numbers on tape.

30  PRINT#5,K

40  NEXT K

50  CLOSE 5           Close logical file #5.
```

In the above cassette example, the data would be accumulated in a 192 character buffer one character at a time. When the capacity of the buffer is exceeded, then data entry is suspended, the tape started, and the buffer contents written to tape. The buffer is initialized to accept up to 192 characters and then the program is allowed to proceed.

NOTE: Not all tape units currently operate with the same START/STOP characteristic as defined for the original tape operating system. In order to obtain reliable operation of the tape recorders, the 192 characters of the buffer should be monitored by the program. Prior to transferring 192 characters, the programmer should turn on the appropriate cassette motor and then wait for at least .1 seconds before transferring the last character.

There are several ways to accomplish this. The simplest is to just POKE 59411,53 for cassette #1 and POKE 59456,207 for cassette #2 after every print statement, this keeps the motor on all of the time and eliminates the problem.

On the other hand, if your programs have time consuming functions like human input, sorting, or other long program run times, you should not run the motor all the time, but obtain the delay either putting a delay loop before each print, or turning the motor off with a POKE 59411,61 for cassette #1 or a POKE 59456,223 for cassette #2 before the long function and turning it back on after it.

8 2 IEEE-488 BUS OUTPUT

The PRINT# command causes BASIC to call an output subroutine which initializes an IEEE-488 device for output. The first step in the command is that the PET reassigns its normal output from the screen device to the physical device that was chosen for the logical file in the open routine. A listen command is sent on the IEEE bus to the physical device and a secondary address specified for that logical file in the OPEN.

BASIC then hands one character at a time to another subroutine which proceeds to transfer that character over the bus with the PET acting as a talker and all addressed devices responding listeners.

When BASIC has finished the PRINT#, another subroutine in the operating system is called and the PET sends an unlisten command to the entire bus and restores the primary address to the screen. This frees the whole bus for the next operation.

This unlisten sequence also sends an EOI signal on the bus, along with the last character sent from BASIC. To accomplish this, each character is stored in a buffer prior to transmission by the IEEE routines and the previous character is sent,

8.2 CMD COMMAND

Normally, each print command deals only with one logical device and at the end of the command the entire bus is unlistened. In some instances, it is advisable to have more than one device on the bus; in order to facilitate this, the special command CMD is provided. CMD is virtually identical to PRINT#, except that at the end of the data transfer, the unlisten routine is not called, thereby, leaving the device on the bus as a listener.

The operating system continues to treat the last device to be commanded by CMD as the primary output device for BASIC. PRINT or LIST commands are then directed to this primary device, rather than to the video screen. More specifically, the CMD of the printer device, followed by LIST, results in a hard copy printed listing, instead of a video screen listing. However, since neither the CMD nor LIST command terminate bus operation for the device, a PRINT# is required to terminate a CMD command.

8.3.1 EXAMPLES OF CMD COMMAND

To list:

```
OPEN 4,4          where 4 is the printer
CMD 4
LIST              will list just the same as the
                  screen, except on the printer.
PRINT#4
```

to print and write to a disc at the same time:

```
* CMD  4          where 4 is open on the printer.
  PRINT#15,A,B,C  where 15 is the floppy disc
```

will result in A,B, and C being stored on the floppy but also being displayed on the printer.

To monitor an input device:

```
**CMD 4              turn on printer
  INPUT#15,A,B,C     read from floppy
```

This will result in the data coming from the floppy being transferred to A , B and C but also being printed as they are being transferred.

* Must be given each time because PRINT# unlistens the bus.
** Need not be given each time, more code may be included between the instructions.

9. CLOSING FILES

Any logical files which have been opened during a program should preferably be closed when no longer required and in the case of tape or disc files, must be closed before the program ends. The following should be kept in mind:

(1) If the total number of logical files currently open exceeds ten, then loss of PET operation will result.

(2) If a logical file assigned to a tape unit is not closed, no 'end of file' mark will be recorded at the end of the physical tape file. If this tape is then loaded into memory, the PET will have no way of knowing the file has ended, and if unwanted and erroneous data is present from a previous recording, it will also be read into memory.

9.1 EXAMPLE OF CLOSE STATEMENT

To close any file, the following simple statement is sufficient:

```
CLOSE logical file
```

If it is required to close logical file number 5, then this becomes:

```
CLOSE 5
```

9.2 TAPE FILE CLOSURE

If a file had been opened on the tape, there are two operations that occur, one an 'end of file' marker is recorded in the next data character, then the tape buffer is forced out onto the cassette.

If during the OPEN the 'end of tape' option was chosen, an 'end of tape file' header block is also forced out on the cassette.

9.3 IEEE-488 NAMED DEVICE CLOSURE

For IEEE-488 devices, which were opened with file names, a special listener command sequence, with the special secondary address of hexadecimal EØ OR'ed with the secondary address from the OPEN is sent. This allows devices such as disc files to be closed by the peripheral controller.

10. ERROR DETECTION: GENERAL

The basic concept of the PET operating system is that the user will be allowed to operate in a free-form format; reading and writing on tapes, discs, printers, in the manner that is most comfortable for him. Because I/O is totally free-form, it is important that the operating system should have a means of informing the user when transmission errors or end of data conditions occur. Sections 10.1 through 10.4 deal with error detection methods available to PET users.

10.1 STATUS WORD (ST)

In order to facilitate INPUT/OUTPUT operation error detection, the PET uses the 'status word' concept in which a byte in memory is manipulated by each of the INPUT/OUTPUT operations for the PET, and can be sampled by the programmer at any time by calling the function ST. Each bit in the status word has a general meaning for all operations and a specific meaning for the individual I/O device. Table 10-1 shows the errors as a function of the ST word value for the tape cassette units, IEEE read/write operations, tape verify and load operations.

ST BIT POSITION	ST NUMERIC VALUE	CASSETTE READ	IEEE R/W	TAPE VERIFY + LOAD
0	1		Time out on write	
1	2		Time out on read	
2	4	Short block		Short block
3	8	Long block		Long block
4	16	Unrecover-able read error		Any mismatch
5	32	Checksum error		Checksum error
6	64	End of file	EOI line	
7	-128	End of tape	Device not present	End of tape.

Table 10-1. Status Word (ST) values correlated with tape cassette, unit and IEEE bus read/write errors.

10.2 ERRORS IEEE DEVICES

There are basically three errors that can occur during an IEEE-488 transfer. First, that the entire bus does not respond to an attention sequence. If this occurs, the IEEE-488 sub-routine sets the device not present (bit 7, or -128). The PET also terminates the current program with a $DEVICE NOT PRESENT ERROR. If the bus responds correctly to the attention, but when the PET goes to write the first character

ST BIT POSITION	ST NUMERIC VALUE	CASSETTE READ	IEEE R/W	TAPE VERIFY + LOAD
0	1		Time out on write	
1	2		Time out on read	
2	4	Short block		Short block
3	8	Long block		Long block
4	16	Unrecoverable read error		Any mismatch
5	32	Checksum error		Checksum error
6	64	End of file	EOI line	
7	-128	End of tape	Device not present	End of tape.

Table 10-1. Status Word (ST) values correlated with tape cassette, unit and IEEE bus read/write errors.

10.2 ERRORS IEE DEVICES

There are basically three errors that can occur during an IEEE-488 transfer. First, that the entire bus does not respond to an attention sequence. If this occurs, the IEEE-488 subroutine sets the device not present (bit 7, or minus 128). If the bus responds correctly to the attention, but when the PET goes to write the first character

to the bus and the phusical device is not present as indicated by having NRFD or NDAC low, the PET, again, gives a 'device not present' indication.

The second error occurs during the process of transferring data to the device, the bus does not respond in the appropriate times and/or if it ceases to respond by means of bringing NRFD and NDAC both high, a write error indication is given in bit Ø. The third error occurs when during read on an IEEE-488 the IEEE device has not sent DAV in less than 65 milliseconds, bit 1 of the status word is sent. . Whenever the EOI line is encountered, the subroutine sets the bit 6 on in the status word and continues to force carriage returns.

10.3 TAPE UNIT ERRORS

The cassette only checks data on read. The errors detected are:

(1) SHORT BLOCK (4). When reading a block from tape, a spacer tone was encountered before the expected number of bytes had been read from that block. Possible cause: attempting to read a short load file as a data record.

(2) LONG BLOCK (8). When reading a block from tape, a spacer tone was not encountered after the expected number of bytes had been read from that block. Possible cause: reading a long load file as data.

(3) UNRECOVERABLE READ ERROR (16). Cause: more than 31 errors on the first block of redundant blocks - or - an error that could not be corrected because it occurred in the same place in both blocks.

(4) CHECKSUM ERROR (32). After a LOAD or reading of data, a checksum is computed over the bytes in RAM and compared to a byte received from the input device. If they do not match, this bit is set.

(5) END OF FILE (64). This bit is set when the end of data file mark is encountered in a tape record.

(6) END OF TAPE (-128). An EOT record was read.

10.4 EXAMPLES OF ST USE

As you can see, there is no status that the PET detects for the writing of tapes, nor errors detected for printing to and reading from the screen. There is an error on writing data out to the IEEE-488 and there is also a series of errors detected on inputting from the IEEE-488 or from tape.

The normal programming technique is to follow an INPUT# or a GET# by either a test or storage of the value of status. Because this is only a single byte of memory and the status changes on each new I/O command, the status is very transient.

```
100  INPUT#2,A
110  INPUT#5,B
120  IF ST=0 THEN 200
```

This code only checks the result of the transfer of data from logical file 5. The results of reading logical file 2 is forever lost. Similarly:

```
100  INPUT#2,A
110  PRINT A
120  IF ST=0 THEN 200
```

In this case, the ST reflects the print status, rather than the results of reading #2.

A correct way to use ST is the following:

```
100  INPUT#2,A,B,C
110  IF ST=0 THEN 200 process normally
120  IF ST=64 THEN 300 (end of data processed with
     no errors).
130  IF ST=2 THEN 400 (time out with no errors)
     each error can now be processed.

140  IF ST AND mask THEN (in which the mask is the
     bit which you are testing for.
```

11. POLLING TECHNIQUES

One technique to poll slow IEEE-488 devices such as sampling devices, which take many minutes to respond, is to use the INPUT# from the device then, if the status indicates time out, process other routines or loop on the INPUT until no error occurs. If there are no errors, the correct data has been finally read and one can process that data information.

By using this sampling technique, a whole series of slow devices can be serviced, along with running a foreground program by use of the real time clock (TI,TI$) and the INPUT/ timeout error sequence to occasionally poll devices.