Integrating HP BASIC with MS-DOS Applications

Edition 1 June 1989

Reorder Number
82301-90022

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Corvallis Information Systems
1000 N.E. Circle Blvd.
Corvallis, OR 97330, U.S.A.

Printing History

Edition 1 June 1989 Mfg. No. 82301-90023

Contents

Appendix B: Keyboard Scancodes

Appendix C: Technical Information About How Multi-Com Works

1

Introduction to the Multi-Com Capability

Introduction

When you install an HP BASIC Language Processor in your computer, you install more than the ability to use HP BASIC — you install a powerful microprocessor similar to the one that runs your personal computer. Until Multi-Com software was developed, it was difficult to use both microprocessors simultaneously, to make both of the computers inside your computer run at the same time.

Background mode — the mode that allows your HP BASIC program to run while you use the MS-DOS operating system to run a second program — is a feature of the HP BASIC Language Processor. Multi-Com software uses this feature and a special communication channel between the BASIC Language Processor and MS-DOS. If you are working in the MS-DOS environment, you can interact with a BASIC program that is running in background mode. You can type in data to be sent to a BASIC program from the MS-DOS environment, or look at data displayed by the BASIC program while it is running in background mode. You may not even need to exit whatever MS-DOS application program you are using to be able to interact with a BASIC program. If you are using Lotus 1-2-3, you can even have your BASIC program send data to the spreadsheet that you are working on!

The communication channel can be used for communication between two or three BASIC Language Processors in the same PC, too. Since up to three language processors can be installed in one PC, a BASIC program running on one language processor can communicate with a BASIC program running on another language processor.

What is Multi-Com?

Multi-Com is a collection of HP BASIC Compiled Subprograms (CSUBs) and one special MS-DOS program that is used for messages. The Multi-Com software is on the "Manual Examples and Selected CSUBs" disk. We'll describe what these CSUBs do in more detail in later chapters, and for a discussion of how CSUBs are used, you should refer to chapter 5 ("Subprograms and User-Defined Functions") of the manual *Programming with HP BASIC*. For now, it is enough to know that the CSUBs included in the Multi-Com software can be called from BASIC programs for the purpose of using the communication channel between one or more BASIC Language Processors and the MS-DOS operating system.

An important feature of the Multi-Com software is the POPCOM window, a window created by an MS-DOS program called POPCOM.COM. This window may be "popped up" or displayed while an MS-DOS application is running, provided the user's display is in alphanumeric (not graphics) mode. The window is used to send information to an HP BASIC program running in background, and to display messages from an HP BASIC program running in background. Although Multi-Com can be used without the POPCOM window, the window allows you to interact with a BASIC program without exiting your MS-DOS application.

The POPCOM window is described in detail in chapter 3.

How Does Multi-Com Work?

The Multi-Com software creates data buffers in an area of memory that is easily accessible by all three BASIC Language Processors and by MS-DOS. These data buffers are, in effect, "mailboxes" that are filled with data that can only be read by a particular BASIC Language Processor or by MS-DOS. The data sent can be ASCII data or codes that are normally associated with the pressing of keys. This means that when you are working in the MS-DOS environment, data sent to your BASIC program can "look" like the pressing of keys. The same is true if you send this type of data from a BASIC program to the MS-DOS environment — it can "look" as if keys on the keyboard were being pressed. There is one "mailbox" for the MS-DOS operating system, and one each for each BASIC Language Processor that is installed.

Full details on how the Multi-Com software works are included in appendix c to this manual.

How Do You Use Multi-Com?

First you must install the Multi-Com software, using the instructions in chapter 2. Next, you should learn all about the POPCOM window by reading chapter 3. If you plan to use more than one BASIC Language Processor, read chapter 4 to learn about multiple language processor considerations.

How you use the Multi-Com software after that depends on your sophistication as an HP BASIC programmer, and on your sense of adventure. Of the four libraries of HP BASIC Compiled Subprograms (CSUBs), one is for beginners, two are for more adventurous programmers, and one is specifically for users of Lotus 1-2-3.

For Beginners

The "POPCOM" Library, called POPLIB, makes the simplest use of the "pop-up" window created by POPCOM.COM. It is described in chapter 5.

For Users Who Want More Direct Control of Communication

The BASIC Communication Library, called BLPLIB, offers more flexibility and control than POPLIB. Principally, the CSUBs in this library are used to send and receive information without the POPCOM window. They are also used to avoid contention for shared resources. Users of BLPLIB must be mindful of multi-task conflicts (described in chapter 4) that may arise. BLPLIB is described in chapter 6.

For Users Who Want Full Control

The Advanced Communication Library, called ADVLIB, offers even more flexibility and control than BLPLIB. The subprograms and functions of this library can be used to control the way the POPCOM window is used, and how the window looks. ADVLIB is described in chapter 7.

For Users Who Want to Use Multi-Com with Lotus 1-2-3

The Lotus 1-2-3 Library, called 123LIB, is included as an example of the way Multi-Com can be used with a specific application. Multi-Com can be used to communicate with many MS-DOS programs, and it can also be used to write custom subprograms, specifically tailored to an MS-DOS application.

123LIB is a library of such custom subprograms. 123LIB provides a convenient way to send commands to Lotus 1-2-3 from HP BASIC, and to move the cell pointer around the 1-2-3 spreadsheet. A source code version of this library is included on the "Manual Examples and Selected CSUBs" disk in a file called S123. It may be used as an example of how you can create custom subprograms for use with Lotus 1-2-3 or other MS-DOS application programs.

For All Multi-Com Users

Chapter 9 contains a sample program that illustrates the way that many of the Multi-Com subprograms and functions can be used.

2

Software Installation

The ability to integrate HP BASIC programs with MS-DOS applications is made available through the Multi-Com files, which are on the "Manual Examples and Selected CSUBs" disk. This section describes how to install these files for use with your applications.

Two types of files provide these capabilities. POPLIB, BLPLIB, ADVLIB, and 123LIB are files that contain HP BASIC Compiled Subprograms (CSUBS). These CSUBS are a library of subprograms and functions, which you may include in your HP BASIC program. For convenience, these files should be located in the same directory as your HP BASIC software.

The file POPCOM.COM is an MS-DOS program that allows the use of a "pop-up" window for communicating with HP BASIC. In addition to being located in a convenient directory, POPCOM.COM must also be executed once to allow "pop-up" window operation. This execution can be done manually when needed or it can be automatically executed at system start-up.

Note

You must have an HP BASIC Language Processor installed in your PC and have already installed the HP BASIC software before continuing! Refer to chapter 2 of the manual *Installing and Using HP BASIC in the MS-DOS Environment* for instructions.

Using the MCINSTAL Program

The easiest method of installation is to use the automatic MCINSTAL program. This program will copy all necessary files and configure your system so that POPCOM.COM is executed each time you boot your computer.

Note

The automatic installation procedure will modify your AUTOEXEC.BAT file by adding a line something like:

C:\BLP\POPCOM.COM

If you do not wish to have your AUTOEXEC.BAT file modified automatically, skip this section and go on to the next, "Manually Installing the Multi-Com Files."

1. Place the "Manual Examples and Selected CSUBs" disk in drive A.
2. Select drive A as the current default drive.
3. When you have the A> prompt, type:

   MCINSTAL [Enter].
4. Follow the instructions on the display. This procedure is very similar to the procedure used when you installed HP BASIC, but simpler. The only information you will need to supply is the directory where you would like the files installed. This should be the same directory you used for the HP BASIC installation.

Manually Installing the Multi-Com Files

You can install all of the Multi-Com files manually using MS-DOS, but you must edit your AUTOEXEC.BAT file if you wish to have the file POPCOM.COM executed when you start your PC. Use the procedure that follows.

1. Use the MS-DOS copy command to copy the following files to the directory (we will assume you named your directory C:\BLP) where you installed HP BASIC:
   - POPLIB
   - BLPLIB
   - ADVLIB
   - 123LIB
   - POPCOM.COM

   As an example,

   COPY A:POPLIB C:\BLP

   copies POPLIB from a floppy disk to the directory C:\BLP on a hard disk.

   Copy all of the files to place the libraries of CSUBs and the file POPCOM.COM into the same directory as BASIC.EXE.

Note

You may choose not to modify your AUTOEXEC.BAT file at all. However, in that case, remember that you must explicitly execute the file POPCOM.COM from the keyboard before you can take advantage of Multi-Com's capabilities.

2. Change to the root directory to modify the AUTOEXEC.BAT file used by your computer. You can use any convenient MS-DOS editor program capable of storing an MS-DOS ASCII file, including Executive MemoMaker, Microsoft Word, and EDLIN. The following line should be added to your AUTOEXEC.BAT file:

   C:\BLP\POPCOM.COM

   AUTOEXEC.BAT must be re-saved as an ASCII file.

3. Reboot your computer or type AUTOEXEC to make sure that the file POPCOM.COM has been executed to allow "pop-up" window operation. You now have all the files you need for Multi-Com capability.

HP BASIC Statements You Will Need to Use with Multi-Com

When you write an HP BASIC program to use Multi-Com, it is helpful to be familiar with the background mode of operation. As is explained in chapter 4 of *Installing and Using HP BASIC in the MS-DOS Environment*, you can place your BASIC Language Processor into background mode by executing the statement:

```
OUTPUT 19;"BACKGROUND"
```

within your program.

You may want to incorporate a particular CSUB or a whole library into your program using a LOADSUB statement. As is explained in chapter 5 ("Subprograms and User-Defined Functions") of *Programming with HP BASIC*, a LOADSUB statement can be used to load one of the CSUBs in a library, or the whole library. For example, the statement:

```
LOADSUB Pop_output FROM "POPLIB"
```

loads the subprogram Pop_output from the library POPLIB. The statement:

```
LOADSUB ALL FROM "POPLIB"
```

loads all of the subprograms from POPLIB. You could then "RE-STORE" the program, thereby incorporating all of the CSUBs of POPLIB into your program.

A third usage of the LOADSUB statement is:

```
LOADSUB FROM "POPLIB"
```

which loads only those CSUBs actually used in your program. Refer to the discussion of LOADSUB in the *HP BASIC Language Reference* for more information on how this keyword is used.

3

The POPCOM Window

The POPCOM window is your window into BASIC from the MS-DOS environment while your BASIC program is running in background. Multi-Com software allows your BASIC program to control the appearance of the window and the way the window is used.

The window itself is created by the MS-DOS program POPCOM.COM. If the file POPCOM.COM is loaded, you can call up the window by pressing the keys [Shift][Ctrl][Backspace].

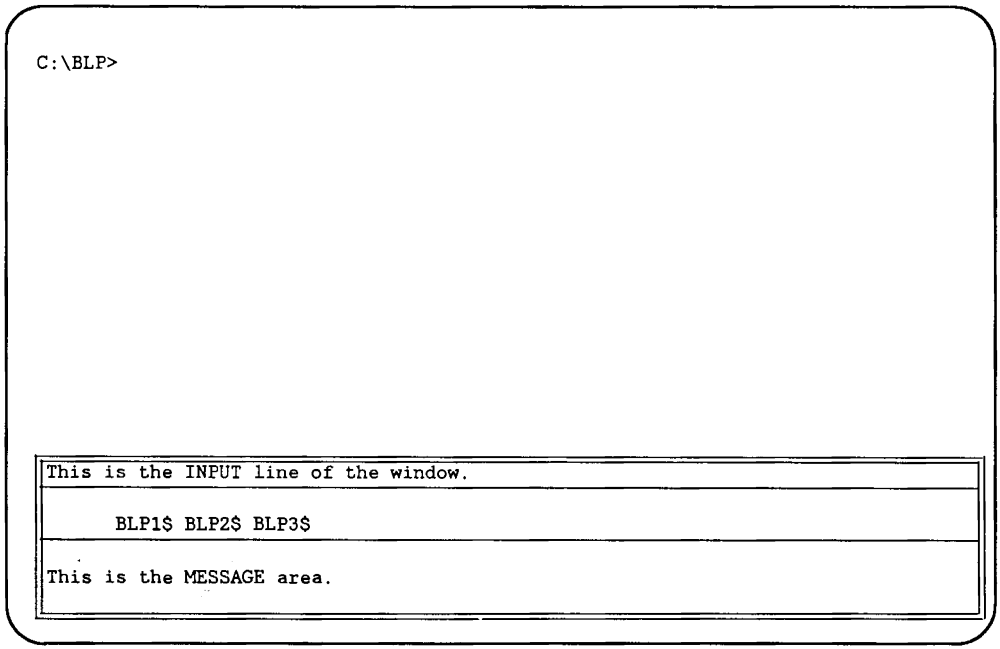
Note

If your MS-DOS application program uses the display in graphics mode as opposed to alphanumeric mode, the POPCOM window cannot be displayed while you are running that MS-DOS program.

If no BASIC program is running in background mode, the window will be displayed, but there can be no communication with BASIC. If a BASIC program that uses the Multi-Com capability happens to be running in background mode, you can use the window to send data to the program from the MS-DOS environment or you can interact with your BASIC program. You may not even have to exit your MS-DOS application to communicate with your BASIC program!

A Typical POPCOM Window

Multi-Com software allows you to change the appearance of the window, but a typical window looks something like this.



The top line of the window is always the input line. It is used to send information to HP BASIC. When BASIC asks a question, this is where you type a response.

The second line of the window is the BLP identification line. Since up to three BASIC Language Processors (BLPs) can be installed in one PC, this line allows you to choose the BLP you want to use. Simply use the cursor keys to select a different BLP.

Note

If other BLPs are not installed, you cannot select them.

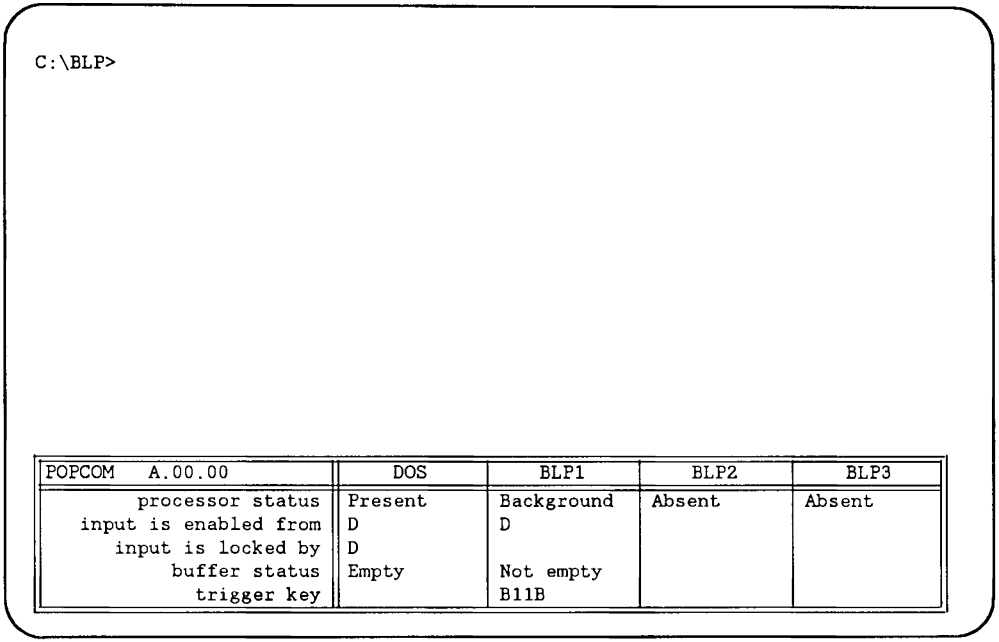
The bottom area is the message area. The size and appearance of this area can be manipulated from a BASIC program to be as small as one line are as large as 18 lines.

When the window is displayed, the following keys have a special meaning:

Esc	Hides the POPCOM window.
Del	Hides the POPCOM window and removes POPCOM.COM from memory. (This only works if there are no BLPs currently in background mode — you must have exited BASIC using Ctrl Break or the BLPs must not be booted. Also, you may not be able to remove POPCOM.COM from memory if you have loaded another memory-resident program since POPCOM.COM was loaded.)
F1	Displays a summary of BLP status.

Interpreting Status Displays

The screen that follows shows a typical status display that might be shown as the result of pressing [F1].



C:\BLP>

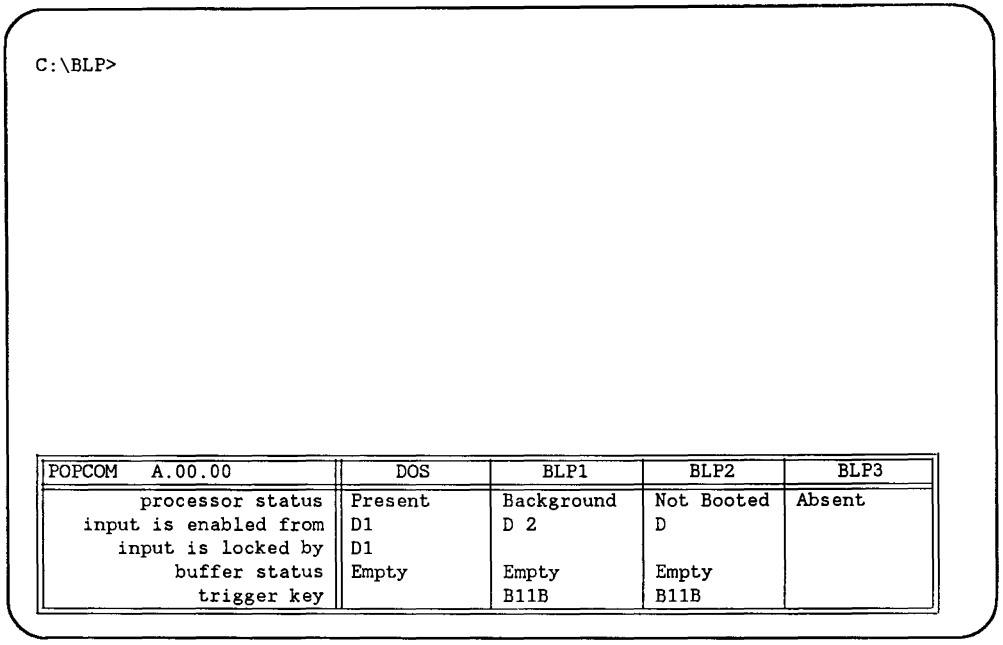
POPCOM A.00.00	DOS	BLP1	BLP2	BLP3
processor status	Present	Background	Absent	Absent
input is enabled from	D	D		
input is locked by	D			
buffer status	Empty	Not empty		
trigger key		B11B		

"Processor status" Line

This line reports one of five conditions for each BLP and for MS-DOS: present, absent, background, not booted, or exited.

"Input is enabled from" Line

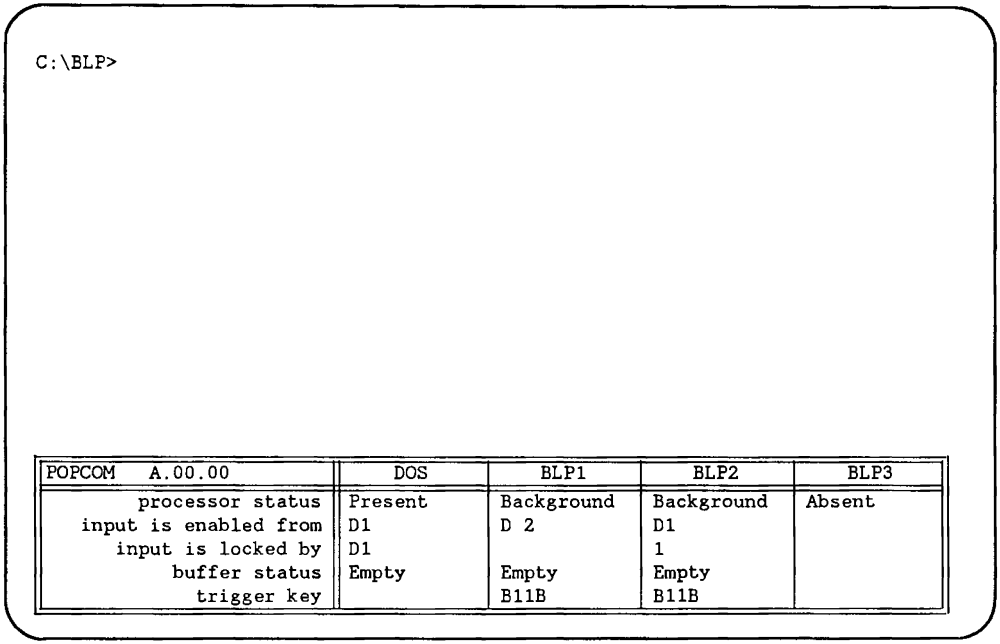
This line provides an indication of which language processors have enabled input from other language processors or DOS. As shown in the screen below, a "D 2" in the BP1 column means that BLP1 will accept information only from DOS and BLP2, but not from BLP3. The "D1" in the DOS column means that DOS will only accept input from itself and BLP1, not from BLP2 or BLP3. The BASIC Language Processors can have input enabled from multiple sources; DOS can have input enabled from itself and, at most, one BLP at a time.



POPCOM A.00.00	DOS	BLP1	BLP2	BLP3
processor status	Present	Background	Not Booted	Absent
input is enabled from	D1	D 2	D	
input is locked by	D1			
buffer status	Empty	Empty	Empty	
trigger key		B11B	B11B	

"Input is locked by" Line

This line provides an indication of which language processors have exclusive access to other language processors or MS-DOS. As shown in the screen below, a " 1" in the BLP2 column means that BLP1 has exclusive access to BLP2. No other language processor may lock BLP2 until BLP1 releases it. The "D1" in the DOS column means that DOS is locked by itself and BLP1. BLP2 and BLP3 will be prevented from accessing DOS. DOS can be locked by itself and one BLP. BLPs can be locked by, at most, one other language processor at a time. For more information about locking and unlocking, refer to chapter 4, "Multiple Language Processor Considerations," and to the descriptions of the function FNBlp_lock and the subprogram Blp_unlock in chapter 6.



POPCOM A.00.00	DOS	BLP1	BLP2	BLP3
processor status	Present	Background	Background	Absent
input is enabled from	D1	D 2	D1	
input is locked by	D1		1	
buffer status	Empty	Empty	Empty	
trigger key		B11B	B11B	

"Buffer status" Line

This line reports whether the buffer for that BLP or for the MS-DOS operating system is empty or not empty.

"Trigger key" Line

This line reports a hexadecimal number representing the HP BASIC keycode that may be sent to the BASIC program if the buffer for that BLP receives data. If the BLP is in ASCII data mode (ready to receive ASCII data rather than keyboard scancodes) and the buffer starts out empty, a trigger keycode will be sent to the BASIC program upon receipt of the first character in the buffer. A trigger keycode is always sent to your BASIC program under these conditions, and your program will respond just as if a key had actually been pressed.

The default trigger keycode is B11B, a hexadecimal value that corresponds to the function key [F1]. When data arrives in a BLP's buffer, the trigger keycode is sent as if you had pressed the key [F1]. If your program does not contain a line with the statement ON KEY 1, BASIC will still interpret the trigger keycode as the [F1] key, and display the typing aid "Edit." No key is actually pressed, but the BASIC program is notified by the trigger keycode that data is in the buffer. No trigger keycode can be used for the DOS buffer, so none is reported in this display.

4

Multiple Language Processor Considerations

Overview

Up to three language processors may be installed in one PC. If your configuration has (or may have in the future) more than one HP BASIC Language Processor, you need to be aware of the considerations described in this chapter. Even if you have only one BASIC Language Processor, you still need to be familiar with requesting exclusive access to MS-DOS using the function FNBlp_lock and releasing it with the subprogram Blp_unlock.

Language processors communicate with each other and with MS-DOS using a special channel. The channel is a resource that is shared. Because of this, you must avoid having two (or more) language processors "talking" to (sending data to) a single destination at the same time. Multi-Com manages this contention by allowing a language processor to gain "exclusive access" to a shared resource.

Gaining Exclusive Access to MS-DOS or another BASIC Language Processor

In order for a language processor to communicate with MS-DOS or another language processor, it must have exclusive access. This is to prevent overlap of information resulting from two or more language processors trying to send data over the channel at once. When one language processor has exclusive access to a resource, other language processors are prevented from gaining access to the same resource. This ensures that only one language processor will be able to "talk" at a time.

If your use of Multi-Com is limited to the six POPLIB functions and subprograms, gaining exclusive access is handled automatically for you. If, on the other hand, you intend to use the functions and subprograms in BLPLIB and/or ADVLIB, your program will have to explicitly gain exclusive access by using the function FNBlp_lock. Even if you have only one language processor, it may not "talk" to MS-DOS unless your program has called FNBlp_lock first. (See the description of FNBlp_lock in chapter 6.)

Releasing Exclusive Access

If your language processor has gained exclusive access to a resource using FNBlp_lock, you should release this resource as soon as you finish sending data to it. This should be done so that another language processor will be able to gain access. Releasing exclusive access is accomplished by calling the subprogram Blp_unlock. (See the description of Blp_unlock in chapter 6.)

Waiting

To attempt to gain exclusive access to a resource, use FNBlp_lock. This function returns with a value of 1 if the resource is available and your language processor is successful in locking it. If, on the other hand, the resource is already locked by another language processor, FNBlp_lock will normally WAIT until it can successfully gain access. As soon as the other language processor releases the resource, FNBlp_lock will perform the lock and return with a value of 1. This is called performing the lock WITH WAIT.

As an option, you can also call FNBlp_lock WITHOUT WAIT. When FNBlp_lock is called WITHOUT WAIT, FNBlp_lock returns immediately, whether the lock was successful or not. If the BASIC Language Processor is successful in gaining exclusive access, FNBlp_lock returns with a value of 1. If the resource is already locked by another language processor, your language processor will not be granted access and the function will return with a value of 0.

Calling FNlp_lock WITH WAIT has the effect of suspending your HP BASIC program until the resource your language processor is trying to "talk" to is available for exclusive use. Calling FNBlp_lock WITHOUT WAIT allows your program to test a resource for availability but then to continue doing useful work if it's not available. Of course, the program will have to test again with FNBlp_lock in order to actually send data to this resource.

If you have more than one BASIC Language Processor, it is to your advantage to release a resource as soon as possible to avoid prolonged waiting by other language processors. Remember, you must call the subprogram Blp_unlock to release any resources that you gained exclusive access to with FNBlp_lock.

As you look through the descriptions of the various functions and subprograms in the following chapters, you will notice several of them can be performed WITH or WITHOUT WAIT. The behavior of these other routines, with regard to Waiting, is exactly the same as it is for FNBlp_lock.

Exclusive Access with the POPLIB Functions and Subprograms

As was mentioned before, the POPLIB routines handle exclusive access requests for you automatically. They do this by temporarily locking the resource they want to "talk" to. When they are finished, they automatically return the exclusive access state to what it was prior to their execution. This means that if your language processor locked a resource using FNBlp_lock prior to a POPLIB operation, it will remain locked. Similarly, if the resource was not locked prior to calling a POPLIB function or subprogram, it will be unlocked once the POPLIB operation is complete.

Grouping Several Multi-Com Operations

If you have more than one language processor, there may be times when you want to insure that a sequence of several Multi-Com functions or subprograms will execute without interruption from the second language processor. You may use FNBlp_lock before the first statement in the sequence and Blp_unlock after the last. In this way, you will prevent another language processor form gaining access to the resource until the entire sequence has been performed.

Memory Considerations

In order to operate in background mode, each BASIC Language Processor needs to have its own memory-resident copy of its operating software. Each copy of this memory-resident program takes up about 100K bytes of PC RAM. If you have more than one language processor installed in your PC, this memory utilization must be considered when determining what PC applications will run concurrently.

5

Using the Library POPLIB

This section documents the POPLIB subprograms and functions that make use of the window created by the MS-DOS file POPCOM.COM. These are the easiest routines of the Multi-Com collection to use. By using these capabilities, the programmer can accomplish two-way communication between the HP BASIC program running in background and a user running a PC application.

Note

If you have more than one BASIC Language Processor, the POPLIB subprograms will protect you from most of the problems associated with sharing resources by automatically locking and unlocking the communication channel.

The functions and subprograms are:

Subprogram Pop_clear	Clears any information in the POPCOM message area
Subprogram Pop_down	Hides the POPCOM window
Function FNPop_enter$	Reads current MS-DOS buffer contents
Function FNPop_input$	Pauses and waits for input from the POPCOM window
Function FNPop_open	Determines if POPCOM.COM is loaded and initializes
Subprogram Pop_output	Sends an ASCII string to POPCOM window

Error codes returned by these functions and subprograms are summarized in appendix A.

Pop_clear

Purpose

This subprogram clears any information in the POPCOM message area.

Usage

```
[CALL] Pop_Clear ([,nowait[,error]])
```

Parameters

nowait	OPTIONAL	0 = WAIT for exclusive window access (default) 1 = Do NOT wait for window access
error	OPTIONAL numeric variable for error return	

Return Variables

none

Example

```
10   CALL Pop_clear
10   CALL Pop_clear (1,Errn)
```

In the first example, Pop_clear waits for exclusive access to MS-DOS. When exclusive access is obtained, information in the window will be erased.

In the second example, Pop_clear does not wait for exclusive access. An error number is returned if exclusive access is not available.

Comments

When more than one BASIC Language Processor is installed in a single PC, the user will need to consider exclusive access and waiting. The default condition is for Pop_output to be performed WITH Wait.

If Wait is not specified, program execution will not be suspended. If access is not available, the Pop_output operation will simply be ignored, and the error return variable will be set to 1.

Successful execution of Pop_clear will cause text in the message area of the POPCOM window to be erased.

Execution of Pop_clear will temporarily lock MS-DOS to the language processor. Upon completion of this operation, the lock/unlock condition will be restored to its previous state. If MS-DOS was locked prior to executing Pop_clear, it will remain locked when the Pop_clear is finished, and if MS-DOS was unlocked prior to executing Pop_clear, it will remain unlocked. See chapter 4, "Multiple Language Processor Considerations," for more information.

Pop_down

Purpose

This subprogram hides the POPCOM window.

Usage

```
[CALL] Pop_down[nowait[,error]]
```

Parameters

nowait	OPTIONAL	0 = WAIT for exclusive window access (default) 1 = Do NOT wait for window access
error	OPTIONAL	numeric variable for error return

Return Variables

Error codes are summarized in appendix A.

Example

```
10  CALL Pop_down
```

In this example, the POPCOM window will be hidden if exclusive access to MS-DOS is obtained. If exclusive access is not available, the program will wait until it is obtained.

Comments

When more than one BASIC Language Processor is installed in a single PC, the user will need to consider exclusive access and waiting. The default condition is for Pop_down to be performed WITH Wait.

When Wait is specified, program execution is suspended until exclusive access to MS-DOS is available. Access is NOT AVAILABLE if another language processor is performing a POPLIB operation, or MS-DOS has been programmatically locked by another language processor.

Execution will continue as soon as the second language processor completes its POPLIB operation or the program unlocks MS-DOS.

If Wait is not specified, program execution will not be suspended. If access is not available, the operation will simply be ignored and the error return variable will be set to 1.

Execution of Pop_down will temporarily lock MS-DOS to the language processor. Upon completion of this operation, the lock/unlock condition will be restored to its previous state. If MS-DOS was locked prior to executing Pop_down, it will remain locked when the Pop_down is finished, and if MS-DOS was unlocked prior to executing Pop_down, it will remain unlocked. See chapter 4, "Multiple Language Processor Considerations," for more information.

FNPop_enter$

Purpose

This function reads the current contents of the MS-DOS data buffer.

Usage

*str$* = FNPop_enter$

Parameters

none

Return Variables

str$ = contents of the data buffer

Example

```
10   DIM A$[77]
20   ON KEY 1 GOSUB 80
30   !
40   LOOP
50      GOSUB Useful_work
60   END LOOP
70   !
80   A$ = FNPop_enter$
90   PRINT A$
100  RETURN
110  END
```

In the example above, lines 40 through 60 are executed repeatedly until Softkey 1 causes the branch specified in line 20. Softkey 1 is the default "trigger key," which is sent to BASIC whenever the MS-DOS buffer has data ready to be read. In this manner, BASIC can do useful work until the user interrupts with a request.

Comments

When FNPop_enter$ is called, the current contents of the data buffer is read. FNPop_enter$ does NOT wait for the user to type anything, and no test is performed for existence of data in the buffer.

FNPop_enter$ does not take into consideration any of exclusive access issues of multiple language processor configurations. If FNPop_enter$ is unable to gain exclusive access, it will simply return a null string. Furthermore, FNPop_enter$ makes no changes to the current locking status of the language processor. See chapter 4, "Multiple Language Processor Considerations," for more information.

FNPop_input$

Purpose

This function is used to request and receive input from the keyboard.

Usage

*str$* = FNPop_input$

Parameters

none

Return Variables

str$ = Data sent from MS-DOS

Example

```
10   DIM A$[77]
20   A$=FNPop_input$
30   DISP A$
40   END
```

Comments

When FNPop_input$ is called, the program is suspended, pending an entry from the user. Data is read, and program execution continues when the user presses the Enter key.

FNPop_open

Purpose

This function allows the user to determine if POPCOM.COM is loaded and initializes the channel.

Usage

variable = FNPop_open [*nowait*[,*error*]]

Parameters

nowait	OPTIONAL	0 = WAIT for exclusive window access (default) 1 = Do NOT wait for window access
error	OPTIONAL	numeric variable for error return

Return Variables

variable = 0 if POPCOM.COM is NOT present
1 if POPCOM.COM IS present

Example

```
10   IF FNPop_open THEN
20      DISP "POPCOM is loaded and ready."
30   ELSE
40      DISP "POPCOM is nowhere to be found.  Please exit HP BASIC and load POPCOM."
50   END IF
```

In this example, FNPop_open is being used to see if POPCOM.COM is currently loaded into the PC's memory. If POPCOM.COM is loaded, the value of FNPop_open is not zero, so the IF condition is met. If POPCOM.COM is not loaded, the ELSE condition will be met.

...FNPop_open

Comments

FNPop_open should be called at least once prior to using any other POPLIB function or subprogram. This is the only way to insure the presence of required MS-DOS software. Unexpected results may be seen if POPCOM.COM is not loaded when calls are made. The default condition is to perform FNPop_open with Wait.

Execution of FNPop_open will temporarily lock MS-DOS to the language processor. Upon completion of this operation, the lock/unlock condition will be restored to its previous state. If MS-DOS was locked prior to executing FNPop_open, it will remain locked when the FNPop_open is finished, and if MS-DOS was unlocked prior to executing FNPop_open, it will remain unlocked. See chapter 4, "Multiple Language Processor Considerations," for more information.

Pop_output

Purpose

This subprogram causes the POPCOM window to be displayed and a character string to be sent to the message area of the window.

Usage

`[CALL] Pop_output (`*str$*`[ ,`*line*`[ ,`*highlight*`[ ,`*nowait*`[ , `*error*`]]]])`

Parameters

str$	String expression to be displayed	
line	OPTIONAL	numeric expression indicating on which line in message area the string will be displayed 0 = Input line 1 = first message line (default) n = nth message line. If n is greater than the maximum number of message lines, the line number will be calculated using the expression: `line = ((n-1) modulo maxlines)+1`
highlight	OPTIONAL	0 = Normal Text (default) 1 = Inverse Video
nowait	OPTIONAL	0 = WAIT for exclusive window access (default) 1 = Do NOT wait for window access
error	OPTIONAL	numeric variable for error return (below)

...Pop_output

Return Variables

error = 0 if operation was successful
non-zero error codes are summarized in appendix A

Examples

```
10 CALL Pop_output("Hello World")
10 CALL Pop_output("Hello World",2,1,1,error)
```

In the first example, the message "Hello World" will be displayed on the first line of the POP-COM window's message area. The message will be displayed without highlight, and Pop_output will wait for exclusive access.

In the second example, the message will be displayed on the second line of the message area, in inverse video. Pop_output will not wait for exclusive access, and an error will be returned if the operation fails.

Comments

When more than one BASIC Language Processor is installed in a single PC, the user will need to consider exclusive access and waiting. The default condition is for Pop_output to be performed WITH Wait.

When Wait is specified, program execution is suspended until exclusive access to MS-DOS is available. Access is NOT AVAILABLE if another language processor is performing a POPLIB operation, or MS-DOS has been programmatically locked by another language processor.

Execution will continue as soon as the second language processor completes its POPLIB operation or the program unlocks MS-DOS.

If Wait is not specified, program execution will not be suspended. If access is not available, the operation will simply be ignored and the error return variable will be set to 1.

Execution of Pop_output will temporarily lock MS-DOS to the language processor. Upon completion of this operation, the lock/unlock condition will be restored to its previous state. If MS-DOS was locked prior to executing Pop_output, it will remain locked when the Pop_output is finished, and if MS-DOS was unlocked prior to executing Pop_output, it will remain unlocked. See chapter 4, "Multiple Language Processor Considerations," for more information.

6

Using the BASIC Communication Library, BLPLIB

The BLPLIB CSUBs provide the HP BASIC programmer a great deal of flexibility. Using these subprograms and functions, the programmer can share HP BASIC data directly with a running MS-DOS application such as spreadsheets or word processor. Users with more than one language processor can use these routines to synchronize control and share data between independent HP BASIC tasks.

The subprograms and functions are:

Function FNBlp_background	Tests to see if the language processor is running in background.
Function FNBlp_disable	Disables input from a BLP or MS-DOS.
Subprogram Blp_enable	Enables input from a BLP or MS-DOS.
Function FNBlp_enter$	Reads current buffer contents.
Function FNBlp_id	Identifies your language processor.
Function FNBlp_input$	Waits for the user to enter information, then reads the buffer contents.
Function FNBlp_lock	Locks MS-DOS or another BLP for input.
Function FNBlp_locked_by	Identifies the language processor that has locked your language processor.
Function FNBlp_more_data	Checks to see if the input buffer is empty.

Function FNBlp_present	Checks for presence of a BLP.
Subprogram Blp_send	Sends message to MS-DOS or another BLP.
Subprogram Blp_unlock	Removes any exclusive input lock.

Error codes returned by these functions and subprograms are summarized in appendix A.

FNBlp_background

Purpose

This function tests to see if the language processor is running in background.

Usage

variable = FNBlp_background

Parameters

none

Return Variables

variable =	numeric variable	0 if in foreground 1 if in background

Example

```
10   !
20   WHILE FNBlp_background
30      GOSUB Do_something
40   END WHILE
50   END
```

In this example, FNBlp_background is used with a WHILE to allow a GOSUB to "do something" only when BASIC is in background mode.

FNBlp_disable

Purpose

This function disables input from another language processor or MS-DOS.

Usage

variable = FNBlp_disable(*BLP#*[,*nowait*[,*error*]])

Parameters

BLP#	0 = MS-DOS 1 = BASIC Language Processor 1 2 = BASIC Language Processor 2 3 = BASIC Language Processor 3	
nowait	OPTIONAL	0 = WAIT for exclusive window access (default) 1 = Do NOT wait for window access
error	OPTIONAL	numeric variable for error return

Return Variables

variable = 0 if disable was successful (default)

Example

```
10   !
20   GOSUB Useful_stuff
30   DISP "I'm done doing useful stuff."
40   FNBlp_disable(0)
50   END
```

Comments

A language processor must be explicitly enabled for input from another language processor or from MS-DOS. Executing FNBlp_disable prevents a designated source from gaining exclusive access with FNBlp_lock.

If no Wait is specified, FNBlp_disable WILL FAIL if your language processor is locked by any other language processor. If Wait is specified, this function will wait for the lock to be removed. Waiting is discussed in chapter 4, "Multiple Language Processor Considerations."

Blp_enable

Purpose

This subprogram enables input from another language processor or MS-DOS and flushes the input buffer.

Usage

```
[CALL] Blp_enable(BLP#[,error])
```

Parameters

BLP# 0 = MS-DOS
1 = BASIC Language Processor 1
2 = BASIC Language Processor 2
3 = BASIC Language Processor 3

error OPTIONAL numeric variable for error return

Return Variables

none

Example

```
10   Blp_enable(0)
20   GOSUB Useful_stuff
30   END
```

In this example, Blp_enable is used to allow input to the language processor from MS-DOS.

Comments

Input must be enabled before any data can be transferred from the designated language processor or from MS-DOS. Input from more than one source can be enabled simultaneously. For example BLP1 could enable input from MS-DOS and BLP2 at the same time. Attempted input from a source that is not enabled will be ignored.

FNBlp_enter$

Purpose

This function reads the contents of the input buffer.

Usage

*str$* = FNBlp_enter$

Parameters

none

Return Variables

str$ = string variable containing the data returned

Example

```
10    DIM A$[77]
20    A$=FNBlp_enter$
30    DISP A$
40    END
```

In this example, FNBlp_enter$ transfers the contents of the buffer to the string variable A$.

Comments

This function is very similar to FNPop_enter$, which is described in the POPLIB documentation. It simply reads the current contents of the buffer. It does not wait for input from the user. It also pays no attention to locking or unlocking issues.

FNBlp_id

Purpose

This function identifies your language processor.

Usage

myid = FNBlp_id

Parameters

none

Return Variables

myid = Numeric variable for language processor number

Example

```
10   !
20   X=FNBlp_id
30   DISP "Hello!  I am language processor "&VAL$(X)
40   !
50   END
```

FNBlp_input$

Purpose

This function is used to request and receive input from the user.

Usage

*str$* = FNBlp_input$

Parameters

none

Return Variables

str$ = Data sent from MS-DOS

Example

```
10    DIM A$[77]
20    A$=FNBlp_input$
30    DISP A$
40    END
```

In this example, FNBlp_input$ transfers the contents of the buffer to the string variable A$. The program will wait on line 20 until a carriage return character is found.

Comments

The function FNBlp_input is similar to FNBlp_enter$, in that it is used to transfer data to the HP BASIC program. It differs from FNBlp_enter$, however, since it does not immediately return the input buffer contents. Instead, it waits until a carriage return character is encountered.

This function is useful when transferring information from applications such as Lotus 1-2-3 that use a carriage return to indicate the end of a spreadsheet row. Be aware, however, that in this situation, it is likely that additional data will be in your input buffer beyond the carriage return. When this happens, the trigger keycode is not re-sent because the buffer never became empty.

In order to ensure that all data is received, FNBlp_more_data should be used in conjunction with FNBlp_input$. See the description of FNBlp_more_data in this chapter.

FNBlp_lock

Purpose

This function is used to exclusively lock a language processor or MS-DOS for input from the calling language processor.

Usage

variable = FNBlp_lock(*BLP#* [,*nowait*[,*error*]])

Parameters

BLP#	0 = MS-DOS 1 = BASIC Language Processor 1 2 = BASIC Language Processor 2 3 = BASIC Language Processor 3	
nowait	OPTIONAL	0 = Wait until lock is successful (default) 1 = do not wait
error	OPTIONAL	numeric variable for error return

Return Variables

variable = 1 if lock is successful

Example

```
10   x = FNBlp_lock(0)
20   Blp_send(0,"This is a message.")
30   PRINT "I just sent a message to MS-DOS."
40   END
```

or

```
10   IF FNBlp_lock(0,1) THEN
20      CALL Useful_stuff
30   ELSE
40      DISP "Sorry, I can't seem to lock MS-DOS."
50   END IF
60   END
```

In the first example, FNBlp_lock attempts to lock input from MS-DOS. If MS-DOS is already locked to another language processor, the program will wait until that lock is released. It can then be locked to this language processor and program execution can continue. It is not necessary to check the return value of the function in this case, as it will only return if the lock is successful.

In the second example, FNBlp_lock is used without Wait. If the lock attempt is successful, the return value of the function is 1, which satisfies the IF condition. If the lock attempt is not successful, the return value of the function is 0, which results in the ELSE condition being met.

Comments

The function FNBlp_lock is used to exclusively lock MS-DOS or another language processor for input from you. It can be invoked with or without Wait. The default is with Wait. A lock request will not be granted if another language processor has the resource currently locked.

If Wait is specified, program execution will be suspended until the lock can be granted. The lock will be granted as soon as no other language processor has the resource locked. The return value of the function will be set to 1.

If Wait is not specified (*nowait*=1), execution will not be suspended if the lock is unsuccessful. If unsuccessful the return value of the function will be set to 0.

FNBlp_locked_by

Purpose

This function identifies the language processor that has locked your language processor.

Usage

BLP# = FNBlp_locked_by

Parameters

none

Return Variables

BLP# = Numeric variable for return of locking BLP number

Example

```
10   !
20   X=FNBlp_locked_by
30   DISP "Oh no!  I'm currently locked by "&VAL$(X)
40   !
50   END
```

FNBlp_more_data

Purpose

This function checks to see whether the input buffer is empty.

Usage

variable = FNBlp_more_data

Parameters

none

Return Variables

variable =	Numeric variable	0 = Buffer empty 1 = Buffer not empty

Example

```
10      DIM A$[77]
20      WHILE FNBlp_more_data
30         A$=FNBlp_input$
40      END WHILE
50   END
```

In this example, FNBlp_more_data is used with a WHILE block to re-execute FNBlp_input$ until all the data has been transferred from the buffer.

FNBlp_present

Purpose

This function tests to see if a certain number language processor is installed.

Usage

variable = FNBlp_present(*BLP#*[,*error*])

Parameters

BLP#	1 = BASIC Language Processor 1 2 = BASIC Language Processor 2 3 = BASIC Language Processor 3
error	OPTIONAL numeric variable for error return

Return Variables

variable =	0 if specified language processor is NOT installed (default) 1 if specified language processor IS installed

Example

```
10   IF FNBlp_present(2) THEN
20      GOSUB Useful_stuff
30   END IF
40   END
```

In this example, FNBlp_present checks to see if BLP2 is installed.

Comments

FNBlp_present is useful as an automatic configuration check and to insure the presence of a language processor.

Blp_send

Purpose

This subprogram sends a character string to MS-DOS or another language processor.

Usage

`[CALL] Blp_send(`*BLP#*`,`*str$*`[,`*error*`])`

Parameters

BLP# 0 = MS-DOS
1 = BASIC Language Processor 1
2 = BASIC Language Processor 2
3 = BASIC Language Processor 3

error OPTIONAL numeric variable for error return

Return Variables

none

Example

```
10   !
20   CALL Blp_send(0,"This is a message for MS-DOS")
30   CALL Blp_send(2,"This is a message for BLP 2")
40   END
```

In this example, lines 20 and 30 send ASCII character strings to MS-DOS and to BLP2, respectively.

Comments

Note

You must have the destination locked before attempting a Blp_send. This is true even if you only have one BLP installed. If you are sending information to another language processor, that language processor must have explicitly enabled input from your language processor. See chapter 4, "Multiple Language Processor Considerations," for more information about locking.

Blp_unlock

Purpose

This subprogram relinquishes exclusive input control over a language processor or MS-DOS.

Usage

```
[CALL] Blp_unlock(BLP#[,error])
```

Parameters

BLP# 0 = MS-DOS
1 = BASIC Language Processor 1
2 = BASIC Language Processor 2
3 = BASIC Language Processor 3

error OPTIONAL numeric variable for error return

Return Variables

none

Example

```
10   CALL Something_useful
20   Blp_unlock(0)
30   END
```

In this example, Blp_unlock is used to release the lock on MS-DOS.

Comments

Blp_unlock is used to relinquish exclusive access to MS-DOS or another language processor. The lock is explicitly cleared without consideration of its state. No Wait is performed.

7

Using the Advanced Communication Library, ADVLIB

ADVLIB is for the BASIC software developer who wishes to take greater control over the appearance and actions of the POPCOM window.

The routines included in this section are:

Subprogram Blp_data_mode	Sets the BLP to ASCII data input mode.
Function FNBlp_dos_idle	Tests to see if MS-DOS is idle or busy.
Subprogram Blp_key_mode	Sets BLP to keyboard scancode mode.
Subprogram Blp_send_key	Sends a keyboard scancode to MS-DOS or a BLP.
Subprogram Blp_set_idle	Sets the MS-DOS idle limit.
Subprogram Blp_trigger_key	Changes default trigger key.
Function FNPop_blp	Tells which BLP the window points to.
Subprogram Pop_lock	Locks the window up or down.
Subprogram Pop_msg_color	Sets the color of the subsequent message.
Function FNPop_msg_lines	Returns the number of message lines available.
Function FNPop_read_buf$	Reads the input buffer.
Subprogram Pop_row	Positions the top of the POPCOM window on the display.

Subprogram Pop_send_buf	Sends an ASCII string to the POPCOM window.
Subprogram Pop_set_input	Sets the number of bytes in user input.
Subprogram Pop_set_lines	Set the number of message lines.
Function FNPop_stat	Returns the display and lock status.
Subprogram Pop_up	Displays the POPCOM window.
Subprogram Pop_unlock	Releases the up/down lock.

Error codes returned by these functions and subprograms are summarized in appendix A.

Blp_data_mode

Purpose

This subprogram sets the input mode of the language processor to expect ASCII data in the buffer.

Usage

```
[CALL] Blp_data_mode
```

Parameters

none

Return Variables

none

Example

```
10  !
20   CALL Blp_data_mode
30  !
40   END
```

Comments

The language processor is capable of accepting input through its buffer as either ASCII data or keyboard scancodes. This routine sets the language processor for ASCII data mode.

FNBlp_dos_idle

Purpose

This function tests to see if MS-DOS is idle or busy.

Usage

variable = FNBlp_dos_idle

Parameters

none

Return Variables

variable =	Numeric variable	0 if MS-DOS is busy 1 if MS-DOS is idle

Example

```
10   !
20   WHILE FNBlp_dos_idle
30      DISP "MS-DOS is not busy"
40      GOSUB Do_something
50   END WHILE
60   END
```

In the example above, the statements on lines 30 and 40 will be executed as long as DOS is idle. As soon as DOS is busy, the value of the function FNBlp_dos_idle becomes 0, and execution halts.

Comments

The BASIC Language Processor monitors MS-DOS to find out whether it is idle or busy. The level of "idleness" is quantified as a number. The more "idle" (available) DOS is, the higher the number is. The subprogram Blp_set_idle allows the user to specify a level of "idleness" to use as a means of deciding whether DOS is idle or busy. The function FNBlp_dos_idle then returns a 0 or a 1, depending on whether the "idleness" reading is greater than or less than the limit set by Blp_set_idle. A 0 indicates that DOS is busy, and a 1 indicates that DOS is idle. Under the same circumstances, different computers will have different levels of "idleness," due to different CPU speeds, RAM speeds, etc.

Blp_key_mode

Purpose

This subprogram sets the input mode of the language processor to expect keyboard scancodes (see appendix B) in the buffer.

Usage

```
[CALL] Blp_key_mode
```

Parameters

none

Return Variables

none

Example

```
10  !
20   CALL Blp_key_mode
30  !
40   END
```

Comments

The language processor is capable of accepting input through its buffer as either ASCII data or keyboard scancodes. This routine sets the language processor for keyboard scancode mode.

Blp_send_key

Purpose

This subprogram sends a keyboard scancode to MS-DOS or another language processor.

Usage

```
[CALL] Blp_send_key(BLP#,key[,error])
```

Parameters

BLP#	0 = MS-DOS 1 = BASIC Language Processor 1 2 = BASIC Language Processor 2 3 = BASIC Language Processor 3
key	Decimal representation of a keyboard scancode (see appendix B)
error	OPTIONAL numeric variable for error return

Return Variables

none

Example

```
10   !
20   CALL Blp_send_key(0,13)  ! Sends RETURN key to MS-DOS
30   END
```

Comments

This function sends keyboard scancodes to the destination. The result is that the destination program thinks that a user typed them on the keyboard.

If you are sending a keyboard scancode to MS-DOS, refer to the list of PC scancodes in the first section of appendix B; if you are sending a message to HP BASIC, refer to the list of HP BASIC keycodes in the second section of appendix B.

...Blp_send_key

No check is made as to the success of this operation. Remember, you must have the destination locked before attempting a Blp_send. Other language processors must have explicitly enabled input from your BLP.

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

Blp_set_idle

Purpose

This subprogram sets the value used to determine if MS-DOS is idle or busy.

Usage

```
[CALL] Pop_set_idle(limit)
```

Parameters

limit Numeric expression of idle limit (default = 8)

Return Variables

none

Example

```
10  !
20   CALL Blp_set_idle(10)
30   OUTPUT 19;"BACKGROUND"
40   WHILE NOT FNBlp_dos_idle
50   END WHILE
60   Blp_send(0,"123")
70   END
```

Comments

The BASIC Language Processor monitors MS-DOS to find out whether it is idle or busy. The level of "idleness" is quantified as a number. The more "idle" (available) DOS is, the higher the number is. The subprogram Blp_set_idle allows the user to specify a level of "idleness" to use as a means of deciding whether DOS is idle or busy. The function FNBlp_dos_idle then returns a 0 or a 1, depending on whether the "idleness" reading is greater than or less than the limit set by Blp_set_idle. A 0 indicates that DOS is busy, and a 1 indicates that DOS is idle.

...Blp_set_idle

Under the same circumstances, different computers will have different levels of "idleness," due to different CPU speeds, RAM speeds, etc. The default limit of 8 seems to work for many PCs. If this value does not produce the desired results, you can try limit values between 2 (for a slow machine) and 20 (for a fast machine).

Blp_trigger_key

Purpose

This subprogram changes the character code of the trigger keycode included with input data.

Usage

```
[CALL] Blp_trigger_key(keycode)
```

Parameters

keycode HP BASIC keycode (see appendix B)

Return Variables

none

Example

```
10  !
20   CALL Blp_trigger_key(F2)
30  !
40   END
```

Comments

Each time data is sent to a language processor, a trigger keycode is sent along with it. This trigger keycode is passed directly to BASIC as an HP BASIC keycode (the decimal value is used) — it is not included in the buffer data. The purpose of the trigger keycode is to allow BASIC to perform interrupt branching when a message is received. The default value for the trigger keycode is function key [F1]. This allows the user to set up an ON KEY 1 interrupt branch to detect incoming messages.

FNPop_blp

Purpose

This function returns the number of the BASIC Language Processor to which the POPCOM window is currently pointing.

Usage

blpid = FNPop_blp

Parameters

none

Return Variables

blpid Numeric variable containing the ID number of the language processor to which POPCOM is "attached"

Example

```
10  !
20   Blpid = FNPop_blp
30   DISP "POPCOM is pointing at BLP # "&VAL$(Blpid)
40   END
```

Comments

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this function is used.

Pop_lock

Purpose

This subprogram causes the POPCOM window to be locked up or down.

Usage

```
[CALL] Pop_lock
```

Parameters

none

Return Variables

none

Example

```
10  !
20   CALL Pop_lock
30   GOSUB Useful_work
40   END
```

...Pop_lock

Comments

Pop_lock prevents the user from displaying or hiding the POPCOM window from the keyboard. If the window is locked up, neither the Esc nor Enter key will hide the window. If the window is locked down, the user is prevented from bringing it up with the Shift Ctrl Backspace keys.

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

Pop_msg_color

Purpose

This subprogram allows the user to set the color of the message lines that follow.

Usage

```
[CALL] Pop_msg_color(color$)
```

Parameters

color$	String expression indicating the attribute bits of the message characters.

Return Variables

none

Example

```
10  !
20   CALL Pop_msg_color("00011111")
30   DISP "The following messages will be bright white on blue."
40   END
```

Comments

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

...Pop_msg_color

The attribute bits are:

Bit	
0	foreground (text) blue
1	foreground (text) green
2	foreground (text) red
3	foreground intensity
4	background blue
5	background green
6	background red
7	blinking text

Combinations of the three RGB (red, green, blue) bits result in these colors:

	Without Intensity	With Intensity
000	black	gray
001	blue	light blue
010	green	light green
011	cyan	light cyan
100	red	light red
101	magenta	light magenta
110	brown	yellow
111	white	bright white

FNPop_msg_lines

Purpose

This function returns the number of lines available for text in the current POPCOM window.

Usage

lines = FNPop_msg_lines

Parameters

none

Return Variables

lines = Number of lines (rows)

Example

```
10  !
20   Lines = FNPop_msg_lines
30   DISP "I can display up to "&VAL$(Lines)&" lines."
40   END
```

Comments

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this function is used.

FNPop_read_buf$

Purpose

This function reads the user input buffer without regard to message termination.

Usage

*str$* = FNPop_read_buf$

Parameters

none

Return Variables

str$ = String variable containing buffer contents

Example

```
10  !
20   %%str$%% = FNPop_read_buf$
30   DISP %%str$%%
40   END
```

Comments

Unlike the other routines available for input from the POPCOM window, this routine does not wait for the user to complete the input line. Bytes are returned as soon as they become available in the buffer. This can be useful for single character input (such as Y or N) or for selectively echoing user input. The message bytes are not removed from the buffer.

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this function is used.

Pop_row

Purpose

This subprogram positions the top of the POPCOM window on the PC display.

Usage

```
[CALL] Pop_row(row)
```

Parameters

line Line number from the top of the PC display.

Return Variables

none

Example

```
10  !
20   CALL Pop_row(7)
30   GOSUB Useful_work
40   END
```

Comments

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

Pop_send_buf

Purpose

This subprogram allows the user to write an ASCII string to the POPCOM window message area.

Usage

```
[CALL] Pop_send(line,msg$,highlight)
```

Parameters

line	Numeric expression indicating which line in the message area the string is to be displayed (0 = message displayed on user input line).	
msg$	String expression containing message to be sent	
highlight	Message highlight	0 = normal 1 = inverse video

Return Variables

none

Example

```
10  !
20   CALL Pop_send_buf("Hello world.",3,1)
30   END
```

Comments

This routine is similar to Pop_output but at a lower level. No checks are made for exclusive access message line parameters being within range.

...Pop_send_buf

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

Pop_set_input

Purpose

This subprogram allows the user to set the maximum number of bytes allowed during each POP-COM input operation.

Usage

```
[CALL] Pop_set_input(bytes)
```

Parameters

bytes	Numeric expression indicating maximum number of characters the MS-DOS user is allowed to input during one operation.

Return Variables

none

Example

```
10  !
20   CALL Pop_set_input(30)
30   DISP "The user can now input up to 30 characters."
40   END
```

...Pop_set_input

Comments

The default input length is 77 bytes. This is also the maximum allowable input length.

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

Pop_set_lines

Purpose

This subprogram allows the user to set the maximum number of lines available for text in POP-COM window.

Usage

```
[CALL] Pop_set_lines(lines)
```

Parameters

lines	Numeric expression indicating number of lines

Return Variables

none

Example

```
10  !
20   CALL Pop_set_lines(4)
30   DISP "I can now display 4 lines to POPCOM windows."
40   END
```

Comments

The absolute maximum number of lines available for text is 18.

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

FNPop_stat

Purpose

This function returns the display and lock status of the POPCOM window.

Usage

stat = FNPop_stat

Parameters

none

Return Variables

stat = Window status bits

0 = Window displayed
1 = Window Hidden
2 = Locked up
3 = Locked down

Example

```
10  !
20   Stat = FNPop_stat
30   IF BIT(Stat,0) THEN DISP "Window is displayed."
40   END
```

Comments

Obviously the window is either displayed or hidden, not both. Consequently the use of bits 0 and 1 is redundant. It is possible for the window to be locked up but be hidden. This means that if and when the window is displayed, the user is prevented from removing it.

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this function is used.

Pop_unlock

Purpose

This subprogram allows the user to show or hide the POPCOM window from the keyboard.

Usage

```
[CALL] Pop_unlock
```

Parameters

none

Return Variables

none

Example

```
10  !
20   CALL Pop_unlock
30   GOSUB Useful_work
40   END
```

Comments

This subprogram has the reverse effect of the subprogram Pop_lock.

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

Pop_up

Purpose

This subprogram causes the POPCOM window to be displayed.

Usage

```
[CALL] Pop_up
```

Parameters

none

Return Variables

none

Example

```
10  !
20   CALL Pop_up
30   GOSUB Useful_work
40   END
```

Comments

Pop_up simply causes the POPCOM window to be displayed. No messages are sent, and there is no test for exclusive access. If MS-DOS is locked by another language processor, the request is ignored.

Note

If you have more than one BLP, unexpected results may occur if this operation is attempted without the destination having been locked beforehand. FNBlp_lock should be used before this subprogram is called.

8

Using the Lotus 1-2-3 Library, 123LIB

123LIB allows the user to easily send to Lotus 1-2-3 the correct character codes to move the cursor around the display and to gain access to the 1-2-3 command line.

Subprogram Command	Sends a 1-2-3 command
Subprogram Down	Moves the cell pointer down
Subprogram Left	Moves the cell pointer left
Subprogram Right	Moves the cell pointer right
Subprogram Up	Moves the cell pointer up

Error codes returned by these functions and subprograms are summarized in appendix A.

Note

These subprograms require Lotus 1-2-3 to be running and in the "READY" mode. Refer to the documentation that came with your copy of Lotus 1-2-3 for an explanation of "READY" mode as well as for other operating instructions.

Command

Purpose

This subprogram sends a command string to Lotus 1-2-3.

Usage

```
[CALL] Command(Cmd$)
```

Parameters

Cmd$ = Command string sent to Lotus 1-2-3

Return Variables

none

Example

```
10  !
20   CALL Command("/ppoouqg")
30   GOSUB Useful_work
40   END
```

Comments

This subprogram is similar to Blp_send, except that it can only send data to MS-DOS, not another language processor.

Down

Purpose

This subprogram moves the Lotus 1-2-3 spreadsheet pointer down.

Usage

```
[CALL] Down(n)
```

Parameters

n = Number of rows to move

Return Variables

none

Example

```
10  !
20      Down(3)
30  !
40   END
```

Comments

No check is made to see if Lotus 1-2-3 is running. This routine simply issues the keycodes for moving the cursor.

Left

Purpose

This subprogram moves the Lotus 1-2-3 spreadsheet pointer to the left.

Usage

```
[CALL] Left(n)
```

Parameters

n = Number of columns to move

Return Variables

none

Example

```
10  !
20      Left(3)
30  !
40   END
```

Comments

No check is made to see if Lotus 1-2-3 is running. This routine simply issues the keycodes for moving the cursor.

Right

Purpose

This subprogram moves the Lotus 1-2-3 spreadsheet pointer to the right.

Usage

```
[CALL] Right(n)
```

Parameters

n = Number of columns to move

Return Variables

none

Example

```
10  !
20      Right(3)
30  !
40   END
```

Comments

No check is made to see if Lotus 1-2-3 is running. This routine simply issues the keycodes for moving the cursor.

Up

Purpose

This subprogram moves the Lotus 1-2-3 spreadsheet pointer up.

Usage

```
[CALL] Up(n)
```

Parameters

n = Number of rows to move

Return Variables

none

Example

```
10  !
20      Up(3)
30  !
40   END
```

Comments

No check is made to see if Lotus 1-2-3 is running. This routine simply issues the keycodes for moving the cursor.

9

A Sample Program Using the Pop-Up Communications Window

The Sample Program

```
1000   DIM A$[256],M$[256]
1010   !
1020   IF NOT FNPop_open(1) THEN
1030      DISP "POPCOM is not loaded.  Exit to DOS and type POPCOM."
1040   ELSE
1050      OUTPUT 19;"BACKGROUND"
1060      WAIT 1
1070      M$="Would you like to continue with the demo?  Please type Yes or No."
1080   !
1090      REPEAT
1100         Pop_output(M$,1)
1110         BEEP 2000,.25
1120         A$=FNPop_input$
1130         A$=UPC$(A$)
1140         M$="You typed: "&A$&".  I said Yes or No.  Please type Yes or No."
1150      UNTIL POS(A$,"Y") OR POS(A$,"N")
1160   !
1170      IF POS(A$,"Y") THEN
1180         Pop_clear
1190         Pop_output("After the window goes away, bring it back and type something.",1)
1200         BEEP 2000,.25
1210         WAIT 3
1220         Pop_clear
1230         Pop_down
1240         ON KEY 1 GOTO A
1250   !
1260         LOOP
1270            FOR I=1 TO 20000
1280            NEXT I
1290         END LOOP
1300   !
1310 A:  !
1320         OFF KEY 1
1330         A$=FNPop_enter$
1340         Pop_output("You typed: "&TRIM$(A$),1)
1350      ELSE
1360         PRINT "DEMO ABORTED"
1370      END IF
1380   !
1390   END IF
1400   !
1410   END
1420   CDEF FNPop_open(OPTIONAL Nowait,Ecode)
1430   CSUB Pop_output(S$,OPTIONAL Lin,Hlgt,Nowait,Ecode)
1440   CDEF FNPop_enter$
1450   CDEF FNPop_input$(OPTIONAL T)
1460   CSUB Pop_down(OPTIONAL Nowait,Ecode)
1470   CSUB Pop_clear(OPTIONAL Nowait,Ecode)
```

Explanation

```
1000   DIM A$[256],M$[256]
1010   !
```

The first thing we do is use FNPop_open to see if POPCOM.COM is loaded. In this case we are calling FNPop_open without Wait and checking the value of the return variable. If the value is 0, we display the message that POPCOM is not loaded. If the value is one, execution continues on line 1050.

```
1020   IF NOT FNPop_open(1) THEN
1030      DISP "POPCOM is not loaded.  Exit to DOS and type POPCOM."
1040   ELSE
1050      OUTPUT 19;"BACKGROUND"
```

BASIC is now running in background. The following WAIT statement simulates the BASIC program doing some useful work until it needs to get the attention of the user. Line 1070 sets up a message string which will be sent to the user.

```
1060      WAIT 1
1070      M$="Would you like to continue with the demo?  Please type Yes or No."
1080   !
1090      REPEAT
```

Pop_output is used to send the message in M$ to the POPCOM window.

```
1100         Pop_output(M$,1)
1110         BEEP 2000,.25
```

FNPop_input$ allows the user to type a response to the message. It waits for the user to press the ENTER key before it transfers the text into the string A$.

```
1120         A$=FNPop_input$
```

A$ is forced to all upper case letters and an error message string is prepared in M$ should our test for yes or no on line 1150 fail.

```
1130         A$=UPC$(A$)
1140         M$="You typed: "&A$&".  I said Yes or No.  Please type Yes or No."
```

If the user's response in A$ does not contain either a "Y" or a "N", the condition tested in line 1150 will not be true. This results in the REPEAT/UNTIL block being re-executed. It will continue to be executed until the user types Y or N.

```
1150       UNTIL POS(A$,"Y") OR POS(A$,"N")
1160   !
1170       IF POS(A$,"Y") THEN
```

Pop_clear is called to erase the contents of the window.

```
1180           Pop_clear
```

Pop_output displays a message, we beep, and wait 3 seconds.

```
1190         Pop_output("After the window goes away, bring it back and type something.",1)
1200         BEEP 2000,.25
1210         WAIT 3
1220         Pop_clear
```

Pop_clear erases the information in the window and Pop_down removes the window from the PC display.

```
1230           Pop_down
```

The ON KEY 1 interrupt condition is established so that BASIC can go on to doing something useful but be alerted when the user types something in the POPCOM window input line. Soft Key 1 is the default trigger keycode for Multi-Com.

```
1240           ON KEY 1 GOTO A
1250   !
```

This loop simulates some useful work being done by the BASIC program. It will continue looping until the user types something on the POPCOM input line and presses [Enter].

```
1260           LOOP
1270              FOR I=1 TO 20000
1280              NEXT I
1290           END LOOP
1300   !
1310 A:    !
```

When the user presses the [Enter] key, the program senses the interrupt and continues execution at this point. We turn off the ON KEY interrupt ability and use FNPop_enter$ to transfer the contents of the input buffer to the string A$. Pop_output is then used to echo what the user typed back to the POPCOM window.

```
1320          OFF KEY 1
1330          A$=FNPop_enter$
1340          Pop_output("You typed: "&TRIM$(A$),1)
1350       ELSE
1360          PRINT "DEMO ABORTED"
1370       END IF
1380  !
1390    END IF
1400  !
1410    END
```

The following CSUBS are from the library POPLIB:

```
1420    CDEF FNPop_open(OPTIONAL Nowait,Ecode)
1430    CSUB Pop_output(S$,OPTIONAL Lin,Hlgt,Nowait,Ecode)
1440    CDEF FNPop_enter$
1450    CDEF FNPop_input$(OPTIONAL T)
1460    CSUB Pop_down(OPTIONAL Nowait,Ecode)
1470    CSUB Pop_clear(OPTIONAL Nowait,Ecode)
```

A

Error Codes

The following error codes are returned from BLPLIB and ADVLIB subprograms and functions:

101	Invalid BLP number (i.e., not 0, 1, 2, or 3).
102	BLP requested is not installed.
103	Attempting to write to a destination without exclusive access.

The following error codes are returned from BLPLIB and ADVLIB subprograms and functions:

201	Invalid BLP number (i.e., not 0, 1, 2, or 3).
202	POPCOM.COM is not loaded.
203	Attempting to write to MS-DOS without exclusive access.

B

Keyboard Scancodes

PC Scancodes

To simulate the "normal" typewriter keystrokes in a MS-DOS application, a background BLP simply writes the desired ASCII code to the DOS data buffer (using a byte access). To simulate the non-typewriter keystrokes (i.e., the softkeys, the cursor keys, etc.), the background BLP must write appropriate scancodes using a word access.

The following table shows the PC scancodes that a BLP must write to the DOS data buffer (using a word access) in order to simulate specific keystrokes. These are listed in ascending numerical order, and are determined by the conventions set by IBM in their ROM BIOS. Any scancode not listed in this table is not supported by the BIOS and, if sent, may cause unexpected results in the MS-DOS application that receives it.

Note

Decimal values of the scancodes are used with subprograms Blp_send and Blp_trigger_key.

If you are using Blp_send to send a keyboard scancode to MS-DOS, refer to the list of PC scancodes first section of this appendix; if you are using BlP_send to send a message to HP BASIC, refer to the list of HP BASIC keycodes in the second section of this appendix.

PC Scancode hex	PC Scancode dec	Keystroke
03	03	Ctrl @
10	16	Alt Q
11	17	Alt W
12	18	Alt E
13	19	Alt R
14	20	Alt T
15	21	Alt Y
16	22	Alt U
17	23	Alt I
18	24	Alt O
19	25	Alt P
1c	28	Enter
1e	30	Alt A
1f	31	Alt S
20	32	Alt D
21	33	Alt F
22	34	Alt G
23	35	Alt H
24	36	Alt J
25	37	Alt K
26	38	Alt L
2c	44	Alt Z
2d	45	Alt X
2e	46	Alt C
2f	47	Alt V
30	48	Alt B
31	49	Alt N
32	50	Alt M

PC Scancode hex	PC Scancode dec	Keystroke
3b	59	F1
3c	60	F2
3d	61	F3
3e	62	F4
3f	63	F5
40	64	F6
41	65	F7
42	66	F8
43	67	F9
44	68	F10
47	71	Home
48	72	up cursor
49	73	PgUp
4b	75	left cursor
4d	77	right cursor
4f	79	End
50	80	down cursor
51	81	PgDn
52	82	Ins
53	83	Del
54	84	Shift F1
55	85	Shift F2
56	86	Shift F3
57	87	Shift F4
58	88	Shift F5
59	89	Shift F6
5a	90	Shift F7

PC Scancode hex	dec	Keystroke	PC Scancode hex	dec	Keystroke
5b	91	Shift F8	70	112	Alt F9
5c	92	Shift F9	71	113	Alt F10
5d	93	Shift F10	72	114	Ctrl numeric pad *
5e	94	Ctrl F1	73	115	Ctrl left cursor
5f	95	Ctrl F2	74	116	Ctrl right cursor
60	96	Ctrl F3	75	117	Ctrl End
61	97	Ctrl F4	76	118	Ctrl PgDn
62	98	Ctrl F5	77	119	Ctrl Home
63	99	Ctrl F6	78	120	Alt 1
64	100	Ctrl F7	79	121	Alt 2
65	101	Ctrl F8	7a	122	Alt 3
66	102	Ctrl F9	7b	123	Alt 4
67	103	Ctrl F10	7c	124	Alt 5
68	104	Alt F1	7d	125	Alt 6
69	105	Alt F2	7e	126	Alt 7
6a	106	Alt F3	7f	127	Alt 8
6b	107	Alt F4	80	128	Alt 9
6c	108	Alt F5	81	129	Alt 0
6d	109	Alt F6	82	130	Alt -
6e	110	Alt F7	83	131	Alt =
6f	111	Alt F8	84	132	Ctrl PgUp

NOTE

Note

Writing a byte value of 13 (decimal) to the DOS buffer will generate a "keystroke" of Ctrl M — this is accomplished by sending a CHR$(13) with the subprogram Blp_send. Writing a word value of 28 (decimal) will generate a "keystroke" of Enter — this is accomplished by calling the subprogram Blp_send_key with a *key* parameter of 28. Both of these have an ASCII code value of 13, but have different scancodes. Some MS-DOS applications will accept both Ctrl M and Enter as carriage return keys, since they only look at the ASCII code portion of the keycode. Others will ONLY accept Enter as a carriage return key, since they look at the scancode as well. Because of this, you should ALWAYS use the Enter code (a word write of 28) to send a carriage return, unless you specifically want to emulate the pressing of Ctrl M.

HP BASIC Keycodes

The following table contains the HP BASIC keycodes that are used as trigger keycodes and as the codes that get written to BLP data buffers when the BLP is in KEYCODE mode (as opposed to the default DATA mode).

Keycode hex	Keycode dec	Function
00	0	unused
01	1	' / ~ (single quote and tilde)
02	2	| / \ (pipe and backslash)
03	3	esc / del
04	4	unused
05	5	break / reset
06	6	stop
07	7	select
08	8	numeric pad enter
09	9	numeric pad tab
0a	10	numeric pad k0 (blank1)
0b	11	numeric pad k1 (blank2)
0c	12	numeric pad k2 (blank3)
0d	13	numeric pad k3 (blank4)
0e	14	home arrow
0f	15	prev
10	16	next
11	17	enter / print
12	18	left extend
13	19	right extend
14	20	system / user
15	21	menu
16	22	clr line
17	23	clr disp
18	24	caps lock
19	25	tab

Keycode hex	dec	Function
1a	26	k0
1b	27	k1 (f1)
1c	28	k2 (f2)
1d	29	k5 (f5)
1e	30	k6 (f6)
1f	31	k7 (f7)
20	32	k3 (f3)
21	33	k4 (f4)
22	34	down arrow
23	35	up arrow
24	36	k8 (f8)
25	37	k9
26	38	left arrow
27	39	right arrow
28	40	insert line
29	41	delete line
2a	42	recall
2b	43	insert char
2c	44	delete char
2d	45	clear->end
2e	46	backspace
2f	47	run
30	48	edit / display functions
31	49	alpha / dump alpha
32	50	graphics / dump graph
33	51	step / any char
34	52	clr line / clear screen
35	53	result / set tab
36	54	prt all / clr tab
37	55	clr io / stop
38	56	pause / reset
39	57	enter (return)
3a	58	continue
3b	59	execute

Keycode hex	Keycode dec	Function
3c	60	numeric pad 0
3d	61	numeric pad . (period)
3e	62	numeric pad , (comma)
3f	63	numeric pad +
40	64	numeric pad 1
41	65	numeric pad 2
42	66	numeric pad 3
43	67	numeric pad -
44	68	numeric pad 4
45	69	numeric pad 5
46	70	numeric pad 6
47	71	numeric pad *
48	72	numeric pad 7
49	73	numeric pad 8
4a	74	numeric pad 9
4b	75	numeric pad /
4c	76	numeric pad E
4d	77	numeric pad (
4e	78	numeric pad)
4f	79	numeric pad ^
50	80	1
51	81	2
52	82	3
53	83	4
54	84	5
55	85	6
56	86	7
57	87	8
58	88	9
59	89	0
5a	90	-
5b	91	=
5c	92	[

Keycode hex	dec	Function
5d	93	]
5e	94	; (semi-colon)
5f	95	' (apostrophe)
60	96	, (comma)
61	97	. (period)
62	98	/ (slash)
63	99	space
64	100	O
65	101	P
66	102	K
67	103	L
68	104	Q
69	105	W
6a	106	E
6b	107	R
6c	108	T
6d	109	Y
6e	110	U
6f	111	I
70	112	A
71	113	S
72	114	D
73	115	F
74	116	G
75	117	H
76	118	J
77	119	M
78	120	Z
79	121	X
7a	122	C
7b	123	V
7c	124	B
7d	125	N
7e	126	left meta
7f	127	right meta

C

Technical Information About How Multi-Com Works

Introduction

To understand the use of the Multi-Com capability, you should be familiar with the structure and operation of the BASIC Language Processor (Revision II) itself. The files used by this version of BASIC are:

HPBLP.SYS	The device driver for the BLP system. This contains usual device driver code, plus three data areas (called "shared memory" or "sharmem," for short), one for each of the three possible BLPs. These shared memory areas provide common storage areas for all of the other system components. Since they are in the device driver, they remain in MS-DOS RAM until the computer is re-booted or turned off, and they are also easily accessible from any of the other programs.
BASIC.EXE	The main control program, which decides what other programs to run, in what order. It contains all of the code necessary for the BACKGROUND mode of operation, and becomes a TSR program (a program that terminates, but stays resident) when you put BASIC into background mode. It has the code that implements the DFS mass storage, the keyboard, timers, Multi-Com, the background "alpha" video, as well as what might be termed the "system control" code.

B1.EXE	The program that boots the operating system into the BLP when "old" boot ROMs are present on the BLP. This is its sole function and it only remains in memory for the duration of the boot process.
B2.EXE	The program that boots the operating system into the BLP when "new" boot ROMs (PC300 BOOTROM) are present on the BLP. This is its sole function and it only remains in memory for the duration of the boot process.
B3.EXE	The "foreground" portion of the BLP software, which contains all of the code necessary to do video (alpha and graphics), HPWLIF mass storage emulation for select code 15, the I/O emulation for PC serial, HP-IB, and GPIO interfaces, and a few other miscellaneous things.
BLP.MSG	An ASCII file containing most of the messages and ASCII strings used in the programs. CAREFUL editing of this file allows translation of most of the BLP messages into localized languages.
BLP.CON	An ASCII file containing some configuration records for modifying the behavior of the BLP system. This file is edited by the CONF.EXE utility.
ACTDISP	The low-level code for the specific display interface in the computer. This file gets loaded by B3.EXE during B3's initialization.
DLIFONT	The file that contains the font used by ACTDISP for the alpha characters. It contains a font for an HP-ROMAN8 character set.
SYSBA513	The BASIC operating system code that gets "booted" into the MC68000's memory on the BLP by either B1.EXE or B2.EXE.

The Multi-Com files include a special DOS file that opens a window to DOS:

POPCOM.COM	A TSR program (a program that terminates but stays resident) that works in conjunction with the device driver (HPBLP.SYS) and BASIC.EXE when in background mode, to provide some of the Multi-Com capabilities.

When you turn on your computer, HPBLP.SYS gets loaded into RAM, because of the line in CONFIG.SYS that says something like:

DRIVER = C:\HPBLP.SYS

At this time, the driver initializes itself and causes the boot ROMs on the BLP to begin executing a selftest.

Later, when BASIC.EXE is run, it opens the device driver in order to access the shared memory, reads its messages out of BLP.MSG, and then, based upon the state of the computer, the BLP, and the command line arguments, runs either B1.EXE, B2.EXE, or B3.EXE. If B1 or B2 is run and completes successfully, then B3 is run, which "boots" BASIC. B1, B2, and B3 are each responsible for reading their own messages from BLP.MSG, as well as reading any of the other files necessary to function.

If you EXIT from BASIC (by pressing CTRL-F10 or executing OUTPUT 19;"EXIT"), then B3 exits back to BASIC.EXE, which then exits back to DOS. If you go into BACKGROUND (by pressing CTRL-F9 or executing OUTPUT 19;"BACKGROUND"), B3 does a little cleanup, passes some information back to BASIC.EXE (about switching into background mode and saving the alpha video contents) and then exits back to BASIC.EXE. BASIC.EXE does a little more cleanup of its own, and then exits back to DOS through the TSR call, so that it stays in memory, using approximately 100K of memory. It's at this point, while BASIC is in background, that the Multi-Com portions become active and particularly useful for communicating with DOS. If Multi-Com is being used to communicate between two or three BLPs, then one of them can be in foreground, with the other(s) in background. But when your BASIC program is communicating with DOS, it has to be in background mode, since DOS is busy running BASIC when BASIC is in foreground mode.

Given that very brief overview of the BLP system, the Multi-Com portion works like this: there is one data buffer, or queue, in the shared memory areas in the device driver for each of the three possible BLPs and one for DOS — a total of four buffers in all. Added to each buffer are a few words of control/status information. Each buffer is used as a transfer area for data that is intended for the associated BLP or DOS. DOS and any BLP can write into any of the buffers (given proper permission), but ONLY the BLP or DOS that is associated with that buffer can read the data from the buffer. So, the buffers act as "mailboxes" for each BLP and DOS. Anyone can put mail in, but only the owner can take it out.

Theory of Operation

Caution

As soon as you start dealing with more than one BLP in a computer at the same time, you run the risk of encountering a deadlock (with respect to the Multi-Com interface). It is because of this that we recommend that the BASIC user stay at the high-level interface to Multi-Com (using the provided CSUBs) if at all possible. You can't harm your computer while using Multi-Com, but you can lock up your computer, requiring a re-boot or power-cycle, resulting in lost data.

Working with MS-DOS Applications

In order to work with most MS-DOS applications, and in order to allow those applications to both send and receive information, it was necessary to implement most of Multi-Com through hooks that were already in MS-DOS. Most DOS programs can write or print to a file, and most programs read input from the keyboard. Since MS-DOS allows you to write to a device driver through the exact same calls as writing to a file, Multi-Com uses already established communication paths.

If a program writes to the device driver, the driver passes that information on to the background BASIC.EXE. BASIC.EXE then inserts the information into the BLP's buffer in the shared memory in the driver, from which a BASIC program can read it (actually, the BASIC program requests it, and BASIC.EXE fetches it from the buffer and passes it "in" to the BASIC program). This allows most MS-DOS applications to send information to a BASIC program. If the BASIC program wants to send information to a MS-DOS application, it sends the information to BASIC.EXE, which puts it in the shared memory buffer in the driver. The driver (at initialization) hooks into INT 16H, the BIOS keyboard function. The next time a request for a key is made through INT 16H by a DOS application, the device driver:

- Gets control before the BIOS does, sees that there is data in the buffer for DOS, then
- Calls BASIC.EXE, which fetches the next character from the buffer, converts it to an INT 16H format scancode, passes it back to the driver, which passes it back to the calling DOS application.

Thus, the data written by the BASIC program looks just like key presses to the DOS application. This is how a BASIC program can send data to almost any MS-DOS application (including COMMAND.COM which is, after all, just another DOS application).

Interrupting BASIC from MS-DOS or Another BLP

When data is written into the buffer for a BLP, it would be useful to have the BASIC program interrupted, to notify it that data is waiting in the buffer. This would keep the BASIC program from constantly polling the buffer. To implement this, we developed the concept of a "trigger keycode". The trigger keycode is a low-level keycode or knobcode that gets sent to BASIC (just as if it were a real key press or knob motion) whenever the buffer goes from an empty state to a not-empty state (i.e., when the first character of data gets written into an empty buffer). If the program has previously executed an appropriate ON KEY, ON KBD, or ON KNOB statement, this will cause a program branch to the specified statement. The trigger keycode is modifiable from the BASIC program, but the default is softkey [F1].

Using Multi-Com with Multiple BLPs

If you have multiple BLPs, you don't want more than one BLP writing to DOS (or to a third BLP) at the same time, or you would end up with gibberish, with the characters of the two messages interwoven with each other. So, we added the concept of "locking." In order for a BLP to send information to DOS, it must first "lock" DOS. Only one BLP can lock DOS at a time. In order for one BLP to send information to a second BLP, the second BLP must have already "enabled for input" the first BLP. Once the first BLP has been enabled-for-input by the second, the first may send information with or without locking the second. This is up to the programmer, and the programmer should beware.

Communicating Between BLPs

One last feature is the ability for one BLP to write keycodes into another BLP's buffer, rather than ASCII characters. These keycodes will then be fed into the second BLP just as if someone were typing from the keyboard. Each BLP determines whether its buffer is receiving keycodes or data characters. It should be obvious that, in a multi-BLP system, using Multi-Com requires a "smart" program on each card, programs that expect their counterparts to be doing specific things and behaving in expected ways.

The Registers

The Multi-Com files make use of BASIC's capability to read and write to absolute memory addresses. These reads and writes can take place in four ways.

The BASIC statement

`READIO (9826, address)`

performs a byte read of the specified address.

The statement

`READIO (-9826, address)`

performs a word read of the specified address.

The statement

`WRITEIO 9826, address; data_byte`

writes a byte of data to the specified address.

The statement

`WRITEIO -9826, address; data_word`

writes a word of data to the specified address.

Refer to the discussions of READIO and WRITEIO in the *HP BASIC Language Reference* for more information on these two keywords.

The table that follows summarizes the READIO/WRITEIO addresses used by the Multi-Com software.

READIO	Address	WRITEIO
SYSTEM STATUS	4358160	DATA TO DOS BUFFER
BLP1 STATUS	4358162	DATA TO BLP1 BUFFER
BLP2 STATUS	4358164	DATA TO BLP2 BUFFER
BLP3 STATUS	4358166	DATA TO BLP3 BUFFER
MYBLP STATUS1	4358168	SET CONTROL
MYBLP STATUS2	4358170	CLR CONTROL
DOS IDLE LIMIT	4358172	DOS IDLE LIMIT
MYBLP BUFFER DATA	4358174	reserved
MYBLP TRIGGER CODE	4358176	MYBLP TRIGGER CODE

Note

The DOS-IDLE-LIMIT register, the buffer data registers, and bits 0 through 7 of MYBLP-TRIGGER-CODE are all numeric values that are read as an entire byte or word.

The SYSTEM STATUS Register

The contents of this status register are summarized in the table that follows.

READIO 4358160 SYSTEM STATUS Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BLP3 Is Present	BLP2 Is Present	BLP1 Is Present	Reserved	BLP3 Has DOS Locked	BLP2 Has DOS Locked	BLP1 Has DOS Locked	Reserved
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
DOS Is Idle	DOS Buff Has Room for Key	DOS Buffer Is Full	DOS Buffer Is Empty	DOS Input Is Enabled from BLP3	DOS Input Is Enabled from BLP2	DOS Input Is Enabled from BLP1	Reserved
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This status register uses word reads only; byte reads are ignored. The upper three bits are used to determine which BLPs are present (not necessarily running). Only one of the DOS-LOCKED bits will be set at a time.

DOS-IDLE-LIMIT is another register in the Multi-Com interface; it stores a value representing the limit of the number of times the keyboard will be checked. The DOS-IS-IDLE bit is true anytime there are more than DOS-IDLE-LIMIT checks of the keyboard within one ten millisecond period.

The purpose of this bit is illustrated in this scenario: the BASIC program goes into background, runs Lotus 1-2-3 (by sending characters to the DOS keyboard through Multi-Com), sends some data to 1-2-3 (again, through the Multi-Com keyboard interface), then exits 1-2-3. Once back at the DOS prompt, it brings BASIC back to the foreground (by sending "BASIC" to the DOS keyboard). The problem is that even after the program tells 1-2-3 to exit, 1-2-3 is still madly buffering the keyboard and will "eat" the first few characters of the "BASIC" command, unless the BASIC program waits a little bit for 1-2-3 to exit. How long? The DOS-IS-IDLE bit serves that purpose. The difficulty lies in discovering the appropriate value to store into DOS-IDLE-LIMIT. This will vary from machine to machine, with different CPU speeds. The default value is 8, but the useful range is from about 5 to 25—use trial and error to determine the best value for your computer. In the above example, we used 1-2-3 for illustration purposes only. The DOS-IS-IDLE bit is not expected to be used a great deal.

The DOS-BUFFER-HAS-ROOM-FOR-KEY bit is used to determine if the DOS buffer has room for a PC scancode. The BASIC program can place both ASCII characters and PC scancodes into the DOS buffer. The ASCII characters take one byte of storage in the buffer, but the scancodes require two. Since these can be intermixed, it's possible for the buffer not to be full, yet for there to be insufficient room for a scancode (i.e., only one empty byte in the buffer). When sending ASCII data to the DOS buffer, the DOS-BUFFER-IS-FULL bit is checked before attempting to write the character to the buffer. But when writing scancodes to the buffer, the DOS-BUFFER-HAS-ROOM-FOR-KEY bit must be checked instead.

The DOS-INPUT-IS-ENABLED-FROM bits are redundant, since they always reflect the state of the DOS-LOCKED bits, but are included for symmetry with the BLP STATUS registers, which can have input enabled without being locked.

The BLP1 STATUS, BLP2 STATUS, BLP3 STATUS, and MYBLP STATUS1 Registers

The contents of these status registers are summarized in the tables that follow.

READIO 4358162 BLP1 STATUS Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0	0	BLP1 Present	1=Key Mode 0=Data Mode	BLP3 Has BLP1 Input Locked	BLP2 Has BLP1 Input Locked	BLP1 Has BLP1 Input Locked	Reserved
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BLP1 Is In Background	BLP1 Is In Foreground	BLP1 Buffer Is Full	BLP1 Buffer Is Empty	BLP1 Input Is Enabled from BLP3	BLP1 Input Is Enabled from BLP2	BLP1 Input Is Enabled from BLP1	BLP1 Input Is Enabled from DOS
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

READIO 4358164 BLP2 STATUS Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0	BLP2 Present	0	1=Key Mode 0=Data Mode	BLP3 Has BLP2 Input Locked	BLP2 Has BLP2 Input Locked	BLP1 Has BLP2 Input Locked	Reserved
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BLP2 Is In Background	BLP2 Is In Foreground	BLP2 Buffer Is Full	BLP2 Buffer Is Empty	BLP2 Input Is Enabled from BLP3	BLP2 Input Is Enabled from BLP2	BLP2 Input Is Enabled from BLP1	BLP2 Input Is Enabled from DOS
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

READIO 4358166 BLP3 STATUS Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
BLP3 Present	0	0	1=Key Mode 0=Data Mode	BLP3 Has BLP3 Input Locked	BLP2 Has Input BLP3 Locked	BLP1 Has Input BLP3 Locked	Reserved
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
BLP3 Is In Background	BLP3 Is In Foreground	BLP3 Buffer Is Full	BLP3 Buffer Is Empty	BLP3 Input Is Enabled from BLP3	BLP3 Input Is Enabled from BLP2	BLP3 Input Is Enabled from BLP1	BLP3 Input Is Enabled from DOS
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

READIO 4358168 MYBLP STATUS1 Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
I Am BLP3	I Am BLP2	I Am BLP1	1=Key Mode 0=Data Mode	BLP3 Has My Input Locked	BLP2 Has My Input Locked	BLP1 Has My Input Locked	Reserved
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
MYBLP Is In Background	MYBLP Is In Foreground	MYBLP Buffer Is Full	MYBLP Buffer Is Empty	My Input Is Enabled from BLP3	My Input Is Enabled from BLP2	My Input Is Enabled from BLP1	My Input Is Enabled from DOS
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

These registers use word reads only; byte reads are ignored. MYBLP STATUS1 is exactly the same as one of the first three. If the program is running on BLP1, then MYBLP STATUS1 will be exactly the same as BLP1 STATUS. The same is true for BLP2 and BLP3. This provides a way for a program to be written generically enough (in most instances) to be able to work on any of three cards without having to worry about which card it's actually running on.

The I-AM-BLPx bits tell you which BLP your BASIC program is executing on, and (when properly shifted and masked) can provide a mask for testing the LOCKED and ENABLED-FOR-INPUT bits in other status registers in a generic way.

KEY/DATA-MODE tells you which mode your buffer is in; the default is DATA. When in DATA mode, only ASCII characters can be written to the buffer. When in KEY mode, only HP BASIC keycodes can be written to the buffer. The default mode is DATA MODE.

If one BLP (let's call it BLP-A) has a second BLP (let's call it BLP-B) locked, BLP-B cannot disable input from BLP-A. If BLP-B hasn't enabled BLP-A for input, BLP-A cannot lock BLP-B.

Either the BACKGROUND bit or the FOREGROUND bit will always be set, but never both, as BASIC.EXE must be running in order for this read to succeed without suspending the 68000 processor's execution.

The default for the INPUT-ENABLE bits is for all BLP inputs to be disabled, and DOS input to be enabled.

The MYBLP STATUS2 Register

The contents of this status register are summarized in the table that follows.

READIO 4358170 MYBLP STATUS2 Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved	Reserved	Reserved	Reserved	I Have BLP3 Input Locked	I Have BLP2 Input Locked	I Have BLP1 Input Locked	I Have DOS Input Locked
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	I Am Enabled for Input to BLP3	I Am Enabled for Input to BLP2	I Am Enabled for Input to BLP1	I Am Enabled for Input to DOS
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This register uses word reads only; byte reads are ignored. These bits are all redundant, as they occur in the other status words for the individual BLPs and DOS. However, this word pulls them all together in one easy location to save the BASIC programmer some work.

The SET CONTROL Register

The contents of this status register are summarized in the table that follows.

WRITEIO 4358168 SET CONTROL Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Send Trigger to BLP3	Send Trigger to BLP2	Send Trigger to BLP1	Set Key Mode	Try to Lock BLP3 Open	Try to Lock BLP2 Open	Try to Lock BLP1 Open	Try to Lock DOS Open
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Flush DOS Buffer	Flush MYBLP Buffer	Enable Input from BLP3	Enable Input from BLP2	Enable Input From BLP1	Enable Input From DOS
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This register uses word writes only; byte writes are ignored. These bits are all active when a 1 is written to them; no action occurs if a 0 is written.

The TRIGGER bits allow one BLP to cause a second BLP's trigger keycode to be sent to the second BLP without actually sending data to its buffer.

In order to lock another BLP or DOS, you must write SET CONTROL with the appropriate bit(s) set to 1. This does not ensure that you will be successful with the lock, however, so a read of MYBLP STATUS2 should be performed to check if you were successful. Once you are successful at locking another BLP or DOS, it will stay locked until you write a 1 in the appropriate location of the CLR CONTROL register.

In order to FLUSH the DOS buffer, you must first LOCK DOS. This can all be accomplished with a single write to SET CONTROL if both the LOCK-DOS and FLUSH-DOS-BUFFER bits are set (LOCK gets done before the FLUSH bit gets checked). However, if the LOCK fails, the flush will not occur.

Note

We discourage use of the FLUSH DOS buffer command, as a general practice since, in most cases, you will be throwing away something that someone wanted to send to DOS.

Note

You can FLUSH a BLP buffer at any time, although this, too, can be a dangerous thing to do (you may lose data).

The ENABLE-FOR-INPUT bits MUST be set before DOS or another BLP can send you data. DOS is enabled by default, other BLPs are not.

The CLR CONTROL Register

The contents of this status register are summarized in the table that follows.

WRITEIO 4358170 CLR CONTROL Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
Reserved	Reserved	Reserved	Set Data Mode	Unlock BLP3	Unlock BLP2	Unlock BLP1	Unlock DOS
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Reserved	Reserved	Reserved	Reserved	Disable Input from BLP3	Disable Input from BLP2	Disable Input from BLP1	Disable Input from DOS
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This register uses word writes only; byte writes are ignored. These bits are all active when a 1 is written to them; no action occurs if a 0 is written.

You can write an UNLOCK command to all BLPs and DOS, even if they're not locked, with no ill effects. If they are locked by you, a write of the UNLOCK bit will ALWAYS unlock them.

You can write a DISABLE-INPUT command to all BLPs and DOS, even if they are not enabled, with no ill effects. If they are enabled for input to you, a write of the DISABLE-INPUT bit will only have an effect if you are NOT locked by that BLP. If you are locked by that BLP, your INPUT-ENABLED status will remain true for that BLP.

The DOS IDLE LIMIT Register

READIO/WRITEIO 4358172 DOS IDLE LIMIT Register. This register uses word reads and writes only; byte reads and writes are ignored. This register has meaning as a single value, rather than as individual bits.

DOS-IDLE-LIMIT stores a value representing the limit of the number of times the keyboard will be checked. The DOS-IS-IDLE bit in the SYSTEM STATUS register is true anytime there are more than DOS-IDLE-LIMIT checks of the keyboard within one ten millisecond period.

The purpose of this bit is illustrated in this scenario: the BASIC program goes into background, runs Lotus 1-2-3 (by sending characters to the DOS keyboard through Multi-Com), sends some data to 1-2-3 (again, through the Multi-Com keyboard interface), then exits 1-2-3. Once back at the DOS prompt, it brings BASIC back to the foreground (by sending "BASIC" to the DOS keyboard). The problem is that even after the program tells 1-2-3 to exit, 1-2-3 is still madly buffering the keyboard and will "eat" the first few characters of the "BASIC" command, unless the BASIC program waits a little bit for 1-2-3 to exit. How long? The DOS-IS-IDLE bit serves that purpose. The difficulty lies in discovering the appropriate value to store into DOS-IDLE-LIMIT. This will vary from machine to machine, with different CPU speeds. The default value is 8, but the useful range is from about 5 to 25—use trial and error to determine the best value for your computer. In the above example, we used 1-2-3 for illustration purposes only. The DOS-IS-IDLE bit is not expected to be used a great deal.

The MYBLP BUFFER DATA Register

The contents of this status register are summarized in the table that follows.

READIO 4358174 MYBLP BUFFER DATA Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
0	0	0	0	0	0	0	0
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
ASCII CODE							
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This register is ignored if MYBLP is in KEYS mode. If it is in DATA mode, a word read will return the next character in the buffer without removing it from the buffer, and a byte read will return the next character from the buffer as well as remove it from the buffer. In other words, a word read is a non-destructive read, a byte read is a destructive read. The word read allows a one-character look-ahead in your data buffer.

The TRIGGER CODE Register

The contents of this status register are summarized in the table that follows.

READIO/WRITEIO 4358176 TRIGGER CODE Register

Bit 15	Bit 14	Bit 13	Bit 12	Bit 11	Bit 10	Bit 9	Bit 8
1	0=Keycode 1=Knob Value	0=Ctrl 1=No Ctrl	0=Shift 1=No Shift	0	0	0	1
Value = 32,768	Value = 16,384	Value = 8,192	Value = 4,096	Value = 2,048	Value = 1,024	Value = 512	Value = 256

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
HP BASIC KEYCODE OR KNOB ROTATION VALUE							
Value = 128	Value = 64	Value = 32	Value = 16	Value = 8	Value = 4	Value = 2	Value = 1

This register uses word reads and writes only; byte reads and writes are ignored. The programmer should BEWARE of using the trigger keycode. The upper bits that are specified as 1 or 0 MUST be ALWAYS written as that value, or the performance of the system may be disrupted. If the KEYCODE/KNOBVAL bit is set to KEYCODE (0), then the lower byte contains an HP BASIC scancode (from the attached table), and the SHIFT and CTRL bits specify the state of those modifier keys when the scancode is sent in to BASIC. If the KEYCODE/ KNOBVAL bit is set to KNOBVAL (1), then the lower byte contains a knob rotation count that is readable by KNOBX or KNOBY, based upon the state of the SHIFT and CTRL bits.

If in KEYCODE mode, then the BASIC program must execute an ON KEY or ON KBD statement in order for the trigger keycode to work. If in KNOBVAL mode, then the BASIC program must execute an ON KNOB statement in order for the trigger keycode to work. Refer to the listing of HP BASIC keycodes in appendix B.

The DATA TO DOS BUFFER

WRITEIO 4358160 Register. Byte writes to this register send a one-byte ASCII code to the DOS buffer. Word writes send a PC scancode to the DOS buffer. The actual scancode that shows up in INT 16H for the ASCII code is what a DOS program would expect; BASIC.EXE uses the ASCII code to look up the scancode in a table, and then passes the two along in the AX register as is normal for INT 16H. When a scancode is written to the DOS buffer, it serves as the value to be placed in the AH register, and a 0 is placed in the AL register before this is returned as the AX value in INT 16H. Refer to your computer's technical reference manual, or to books about the PC BIOS for more information about INT16H. PC scancodes are listed in appendix B.

The DATA TO BLPx Buffers

WRITEIO 4358162 DATA TO BLP1 Buffer
WRITEIO 4358164 DATA TO BLP2 Buffer
WRITEIO 4358166 DATA TO BLP3 Buffer

If BLPx is in DATA mode, then word writes are ignored and byte writes are ASCII data. If BLPx is in KEY mode, then byte writes are ignored, and word writes are keycodes which must conform to the format described for TRIGGER CODE (above).

Index

T

U

W