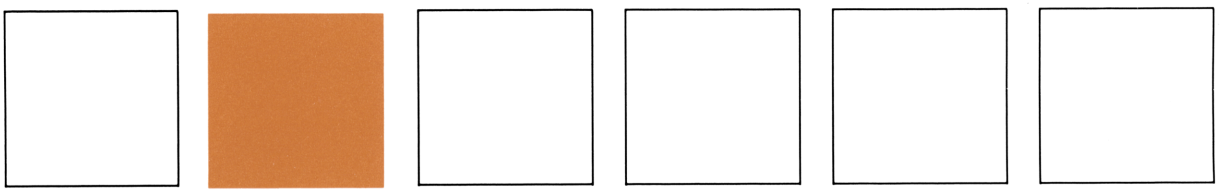
HEWLETT PACKARD

HP 82478A

Debugger

Owner's Manual

For the HP-71



HP 82478A

Debugger

Owner's Manual

For the HP-71

July 1986

82478-90001

Printed in Singapore

Notice

Hewlett-Packard Company makes no express or implied warranty with regard to the keystroke procedures and program material offered or their merchantability or their fitness for any particular purpose. The keystroke procedures and program material are made available solely on an "as is" basis, and the entire risk as to their quality and performance is with the user. Should the keystroke procedures or program material prove defective, the user (and not Hewlett-Packard nor any other party) shall bear the entire cost of all necessary correction and all incidental or consequential damages. Hewlett-Packard Company shall not be liable for any incidental or consequential damages in connection with or arising out of the furnishing, use, or performance of the keystroke procedures or program material.

Introducing the Debugger for the HP-71

The HP-71 debugger offers programmers the ability to simulate assembly language on the HP-71B handheld computer.

If your programming experience is in BASIC, you may be drawn to assembly language by its promise of greater speed and flexibility. Software development at this level requires the same approach to inspection and proof — the tempering of logic often referred to as the debugging stage. In assembly language this process is much more focused since execution flow and register usage is at the lowest level, in the Central Processing Unit (CPU), where the built-in protections afforded by an operating system are not available. In many cases the best method to certify programming logic is by direct testing. You need a tool which allows you to follow execution and register contents at each step — a debugger.

In tandem with the FORTH/Assembler ROM or with PC-based Saturn Assemblers, the debugger provides an accurate, affordable, and portable tool for assembly language development on the HP-71. Its many options allow you to customize its operation, even midway through a debugging session. Its modularity is designed to allow the addition of more features through assembly language enhancements.

This manual provides a description of the file setup, CPU emulation, output, and register manipulation. For details on HP-71 file structure, keywords, and Independent RAMs (IRAMs), refer to the *HP-71 Owner's Manual*.

Contents

How to Use This Manual

Sections 1 and 2 of this manual present basic characteristics and limitations of the debugger system. All programmers should be familiar with these principles.

First-time users of the debugger should read section 3, "Setting Up the Debugger." This section will show you how to prepare the files, how to enter and exit the debugger system, and how to examine the emulated CPU registers.

Important information on the input and display features of the debugger is contained in section 4, "Using the Keyboard," and section 5, "Register Details." You need this information to set up and interpret registers during a debugging session. These two sections are central references for all users of the debugger.

After you are familiar with the structure of the debugger system, you will be ready to perform actual debugging. To do this, you need to know how to run the emulator and how to interface to BASIC. These are the topics of section 6, "The Emulator," and section 7, "Back and Forth from BASIC."

If you plan to use an HP-IL display device or remote keyboard for a more productive debugging session, read section 8, "Using HP-IL."

Users familiar with the debugger system, or those interested in customizing the debugger software, should read section 9, "Additional Features and Operating Details." This section describes advanced interfaces to assembly language routines and techniques to use when emulating in the HP-71 operating system.

For reference information about the BASIC keywords in the debugger, refer to appendix B, "BASIC Keywords."

Throughout this manual, keystrokes are indicated as follows:

- unshifted keys: [K] , [5] , [RUN] , etc.
- shifted keys: f[DELETE] (the f-shifted [M] key),
  f[EDIT] (the f-shifted [Z] key),
  g["] (the g-shifted [2] key),
  g[CMDS] (the g-shifted [ENDLINE] key), etc.

For clarity, the g-shifted keystrokes for characters "less-than" ("<") and "greater-than" (">") are shown as g[<] and g[>], respectively. The four arrow keys are shown as [←], [→], [↑] and [↓]. The four corresponding g-shifted arrow keys are g[←], g[→], g[↑] and g[↓].

Section 1

Assembly Language Development on the HP-71

Programming with assembly language on the HP-71 should be attempted only after thoroughly understanding the characteristics of the operating system. The *HP-71 Software Internal Design Specifications* (IDS), Volumes I through III, are available to help programmers understand and work with the computer. The IDS part numbers for HP-71 version 1BBBB are HP 00071-90068, 00071-90069, and 00071-90070. Determine the version of your HP-71 from the VER$ function.

Caution

Since assembly language controls the computer at the CPU level, you have access to memory contents without the built-in protections of the BASIC operating system. Although damage to the computer itself, as well as any peripherals, is unlikely, files and data in memory can be easily corrupted. As protection against Memory Lost, you should copy any important files to a mass memory device before working with assembly language or the debugger. Without the overhead of the operating system, the assembly language programmer assumes responsibility for the integrity of memory.

To use the debugger, you should be familiar with section 6 of the *HP-71 Owner's Manual*, "File Operations." You need to know how to copy files between memory devices. In particular, you should understand the creation and use of Independent RAMs.

The HP-71 Software IDS provides details on memory usage, file structure, internal data storage, and interfaces to assembly language routines. This information is required for virtually all software development. Chapter 16, IDS Volume I, is indispensable when working with the debugger. It describes the Saturn CPU instruction set and some of the CPU and bus architecture.

The HP-71 Hardware IDS (part number HP 00071-90071) may also prove valuable, as it goes into more detail on the CPU and bus. Chapter 3 of that document also shows the CPU instruction set. If you plan to use the HP-IL module extensively in assembly language applications, you may also need the HP-IL IDS (part number HP 82401-90023).

You need to be familiar with the source document for your assembler, for debugging often involves frequent code changes and reassemblies. Typically, this will be the *HP-71 FORTH/Assembler ROM Owner's Manual*, part number HP 82441-90001. That document also lists the CPU instruction set and capabilities.

Section 2

The Debugger System

The debugger is a RAM-based program of less than 12K bytes which resides in HP-71 memory along with the target assembly language. It is designed in three modules, comprising four files which can be copied into and out of memory as desired.

Before using the debugger, you should write-protect the original medium, make copies of the files onto another disk or cassette, and keep the original in a safe place for a back-up copy. This will offer protection against inadvertent changes to the debugger file contents in RAM.

Once the debugger is copied into HP-71 memory, no other equipment is necessary to run the debugger. You need to have enough RAM to read in the debugger files (from 8K to 12K, depending on your configuration), preferably set up as Independent RAM (IRAM). If this does not leave enough memory for your assembly language application, you may need to purchase more plug-in memory modules.

The debugger is compatible with the HP-71 operating system, and may reside in memory without affecting the operating characteristics of the computer. Activating the debugger places the HP-71 operating system in a state of suspension, where individual CPU registers and memory locations can be examined and edited. Assembly language routines can be emulated with interrupt, single-step, and break point capability. The HP-71 keyboard is used to direct the debugger, and output may be sent to both the LCD and an HP-IL display device.

Debugger Files

The HP-71 debugger system consists of four LEX files:

1. Main file DBGMAINA: a LEX file which contains the keywords and controlling routines for accessing the debugger from BASIC. It also responds to the VER$ function, and generates the errors and warnings when using the keywords.
2. Core files DBGLEX1A and DBGLEX2A: two LEX files which contain the core of the debugger system, including RAM storage, emulator, and input/output routines.
3. Options file DBGLEX3A: a LEX file which provides additional enhancements and options to control output from the debugger.

Debugger Version

VER$ String. The version of the debugger that you are running can be determined from the VER$ string, which returns a value such as "DBG:A". The version designation appears as part of each file name — in the names above, it is the "A" as the last character. If you are running version "DBG:B" of the debugger, you will be working with files DBGMAINB, DBGLEX1B, and so on. Throughout this manual, the files are referred to with the "A" designation, even though you may have a later version.

Compatibility. If revisions to the debugger are published with subsequent VER$ designations, they will be incompatible with previous versions. For example, if you have version "DBG:B", you cannot run the debugger with the DBGLEX2A file. The debugger will issue an error if you try to activate it with unmatched files, as they are encoded internally with the VER$ designation.

Debugger Files in Memory

Not all of the debugger files need to be copied into memory to run the debugger. The core module files (DBGLEX1A and DBGLEX2A) are the only ones necessary to work through a debugging session. DBGMAINA is required to activate the debugger, but not to continue a debugging session. File DBGLEX3A is optional.

The core and options modules are meant to be copied into and run from IRAM. This gives them relatively fixed locations for storage of key registers. The main file DBGMAINA may be run in MAIN RAM.

The Purpose of Each File. File DBGMAINA contains the necessary keywords to set up and activate the debugger. It must be in memory in order to run the debugger, but once activated, DBGMAINA may be purged if the additional room is required for other applications. However, if you deactivate the debugger, you will need to copy DBGMAINA back into memory to access the system again.

With DBGMAINA in memory, the following features are available:

- The debugger's response to the VER$ function ("DBG:A").
- Three BASIC keywords:
  - — **DEBUG** (and its variant **DEBUG** *);
  - — **DBGADDR$**, for ascertaining base addresses for the DBGLEX files;
  - — **DDISPLAY IS**, to set up the debugger HP-IL display device.

In addition, the errors and warnings associated with the keywords are generated by the DBGMAINA file. The keywords are explained below and in appendix B, "BASIC Keywords."

Files DBGLEX1A and DBGLEX2A are required to run a debugger session. They contain the key detection and display routines, register manipulation and editing capabilities, as well as RAM storage and CPU emulator.

File DBGLEX3A may be omitted to increase the amount of memory available to your application. When present, it provides the following features:

- Disassembler, to display opcode mnemonics.
- 80-column interface for the HP-IL display device.
- Adjustment of displayed addresses to match listings.
- Extended command menu.

System Limitations

Running the debugger out of IRAM permits it to use relatively fixed RAM locations for its buffer requirements. However, this makes it vulnerable to low-level configuration commands which de-address chips, so by necessity there are a few things the debugger cannot emulate. It is expected that fewer than 2% of debugger applications will encounter this limitation. In addition, since the

debugger's emulator runs at about 1/75 real speed, critical timing loops in assembly language must be adjusted accordingly. Aside from these cautions, the debugger can handle almost any assembly language application.

The debugger is particularly vulnerable to the precise thing it is used to uncover — bugs in assembly language code. Because it resides in RAM, writing data to any address within its file boundaries may render the debugger inoperable. Thus, encountering a bug which clears or moves a large chunk of memory will likely corrupt the debugger on its way to a Memory Lost. For this reason, you should always have a write-protected backup copy of the debugger available on disk or other medium.

Operating Precautions

When you have activated the debugger in your HP-71, there are some actions you should never attempt without first deactivating the debugger. These include:

1. Removing or installing a plug-in module.
2. Executing FREE PORT or CLAIM PORT.
3. Removing and installing batteries.
4. Copying, purging, or otherwise moving a DBGLEX file.

Performing any of the above actions with the debugger active may disrupt the operation of the HP-71, causing a Memory Lost condition or requiring an INIT: 3 to recover control of your computer.

The operating manual for each HP-71 plug-in device contains directions for installing and removing the module. In addition to deactivating the debugger, you must observe the precautions for inserting plug-ins in the HP-71 ports.

Specific to HP-71

The debugger is HP-71 specific; that is, it will not run on other Saturn CPU systems. Several mainframe utility routines and RAM pointers are used in the debugger, and they are expected to reside at the addresses found in all versions of the HP-71.

The emulator will run on all versions of the HP-71, and is designed for complete compatibility with the Saturn CPU instruction set published in Chapter 16 of the IDS, Volume I. Later versions of the Saturn CPU may be used in HP-71s with serial numbers above 2623A00000, and may contain new additional opcodes. Although the resident operating system will not use these new opcodes, custom assembly language routines might be written which do. The debugger will not emulate them properly. (Because the emulator is RAM-based, advanced users can change or add new instructions by changing the DBGLEX files. Guidelines for this are given in section 9, "Additional Features and Operating Details.")

Equipment Needed

No extra equipment or peripherals are needed to run the debugger in the HP-71. However, debugging is easiest when you use an HP-IL display device — which can be a printer — for viewing registers and emulator results. This requires the HP-71 HP-IL interface and a display device or printer. Not all debugging sessions can run with the external display device, however.

Refer to section 8, "Using HP-IL," for details.

For the safest use of an external display, you should use the HP-71 Dual HP-IL Adapter (part number HP 82402A). This will allow you to dedicate one loop for the debugger to display registers, while the other loop can be used by the HP-71 operating system in emulation.

You may also use a KEYBOARD IS device in some debugging situations. This requires an HP-IL module, a remote keyboard, and the LEX file "KEYBOARD", available from the HP-71 User's Library or found in the FORTH/Assembler ROM and HP-IL LINK. Section 8, "Using HP-IL," has more information regarding the use of Dual HP-IL and a remote keyboard.

Using peripherals with the debugger requires a certain knowledge of HP-IL. You should become familiar with operation and control of the loop by consulting the *HP-IL Interface Owner's Manual*.

You may also find that using a calculator with hexadecimal capability will greatly simplify address calculations. In most cases, you cannot interrupt a debugging session to have the HP-71 perform these calculations.

Section 3

Setting Up the Debugger

The debugger is invoked from BASIC, after you have copied the LEX files into memory. This section describes the steps needed to prepare for a debugging session, as well as an overview of the main feature of the system — the stack structure.

File Setup

The debugger LEX files are provided on mass media, which you should keep as a write-protected backup. In preparation for a debugging session, you need to copy DBGLEX1A and DBGLEX2A into memory as outlined below. The options module, DBGLEX3A, is needed only to access several enhancements for debugger output.

Copying DBGLEX3A into memory provides the following capabilities:

- A disassembler, to display opcode mnemonics.
- An 80-column interface for the HP-IL display device.
- Adjustment of displayed addresses to match listings (BIAS).
- An extended command menu.

This is the recommended procedure for loading the debugger LEX files:

1. Use the FREE PORT statement to create IRAMs. Unless you have a plug-in RAM module larger than 4K, you will need to free one port for each DBGLEX file. If you have larger RAMs, you can copy two or all three DBGLEX files into a single IRAM (provided DBGLEX1A is the first file in the IRAM). The HP-71B has more than 16K of RAM built-in, so you do not need plug-in memory modules unless your assembly language routines require more memory.
2. Copy DBGLEX1A and DBGLEX2A, and, if desired, DBGLEX3A into the IRAMs. The file DBGLEX1A *must* reside at an address ending with "008". Copying it into an IRAM as the first (or only) file guarantees this to be the case. DBGLEX2A and DBGLEX3A may reside at any address. All three files may be put in memory in any order.
3. Copy DBGMAINA into MAIN memory (or IRAM if desired).

For example,

```
FREE PORT(0)                          ! Create one IRAM for
FREE PORT(.01)                        ! each DBGLEX file.
FREE PORT(.03)
COPY DBGLEX1A:TAPE TO :PORT(.03)
COPY DBGLEX2A:TAPE TO :PORT(.01)
COPY DBGLEX3A:TAPE TO :PORT(0)
COPY DBGMAINA:TAPE
```

Now the debugger is ready to run. Whether the target assembly language is a mainframe routine,

a LEX file, a BIN file, or a FORTH primitive created by the FORTH/Assembler ROM, you can activate the debugger and start emulating the code.

Debugger Display Device. With the file DBGMAINA in memory and the HP-IL interface plugged in, the BASIC statement DDISPLAY IS can be used to specify a display device for debugger output. If you have a display device on the loop, you can direct debugger registers to HP-IL for a more productive debugging session. Details on using this statement are in section 8, "Using HP-IL."

It is not necessary to use HP-IL for debugger displays. In fact, in some debugging situations you should avoid doing so. The debugger defaults to using the LCD only, if DDISPLAY IS is not specified. Until you are familiar with the keyboard interface and register structure, it is advisable to use the LCD as the default display device.

Activating and Deactivating the Debugger

Activating the debugger implies that the built-in protections of the BASIC operating system are left behind, even when you temporarily suspend the debugger to drop into the BASIC environment.

With the debugger active, there are some actions you should never attempt without first deactivating the debugger. These include:

1. Removing or installing a plug-in module.
2. Executing FREE PORT or CLAIM PORT.
3. Removing and installing batteries.
4. Copying, purging, or otherwise moving a DBGLEX file.

Activate with DEBUG Statement. You activate the debugger with the **DEBUG** statement (or its variant, **DEBUG** *) provided in the DBGMAINA file. The **DEBUG** statement is described below.

Deactivate with RECOVER Sequence. You deactivate the debugger with the **RECOVER: 2** or **RECOVER: 3** sequences. These are described below.

Entering the Debugger

The file DBGMAINA must be in memory to activate the debugger, as it contains the **DEBUG** statement. Once activated, DBGMAINA can be purged from memory if the additional space is required for your application. However, if deactivated, you will need to copy DBGMAINA into memory to activate the debugger again.

To become familiar with the debugger system, perform the following three steps. The subsequent paragraphs explain the actions in detail.

1. Activate the debugger:*

Keystroke	Display
DEBUG [ENDLINE]	PC:00000: 2034EE100060

   The debugger system is now ready for viewing registers or emulating code. However, be careful not to press any keys other than those described below. You will be learning how to control the debugger shortly.

2. Drop into the BASIC operating system, momentarily suspending the debugger:

Keystroke	Display
[Q][Q] (press [Q] twice, within 1½ seconds)	> (the BASIC prompt)

   You are now back in the BASIC environment, with the debugger still in charge of the HP-71. (The alarm annunciator ((•)) indicates this.) The full BASIC operating system is available. You can execute "BEEP", "RESTORE IO", "CAT ALL", or run programs, etc.

3. Deactivate the debugger. Press the [ON] and [/] keys *simultaneously*; instead of the familiar **INIT: 1** prompt, you will see the debugger prompt **RECOVER: 1**.

Keystroke	Display
[ON][/]	RECOVER: 1
[2]	RECOVER: 2
[ENDLINE]	> (the BASIC prompt)

   The debugger is deactivated, and the BASIC operating system has regained complete control over the computer.

The DEBUG Statement

The BASIC statement **DEBUG** is used to first enter the debugger. Executing **DEBUG** displays the debugger's PC (CPU Program Counter) register. It is the PC which points to the opcode for emulation, so the address in this register is crucial to further debugging.

---

* If the **DEBUG** statement causes an error, you need to check your file setup. Directions for copying in the debugger files are given at the beginning of the section.

A discussion of register usage and editing follows in section 5, "Register Details."

Register Contents. When first activated, some debugger control registers need to be set to the address of the target assembly language. Typically, you set a debugger Break Point register to the address of your target routine, allow the operating system to execute within the vicinity of the routine, then direct the debugger to take over. It will halt at the break point you set, where you can examine registers or continue execution with the emulator. This process is described in the succeeding sections.

> **Example:** Suppose you have a BIN file "BINTEST" which you have assembled with the FORTH/Assembler ROM. Before entering the debugger, execute the following:
>
> ADDR$("BINTEST")
>
> to get the address of the code. Although this is the address of the file header, it is in the general area of the code you will be debugging. Using this address and your assembler listing, determine the address of the target routine. You need to remember this value for later entry into a Break Point register.

Other than possibly specifying an HP-IL display device, there is no initialization required when activating the debugger. All registers are cleared when the debugger is activated for the first time. Subsequently, executing **DEBUG** causes registers to be restored to their contents from when the debugger was last exited.* Since occasionally you will want to reset the registers, there are methods to clear them or re-initialize the entire system. These are explained in section 4, "Using the Keyboard."

The DEBUG * Statement

An alternate form of the **DEBUG** statement is **DEBUG** * which causes the debugger to take over and begin emulation immediately. It sets up its registers for an exit through the mainframe entry point "NXTSTM" (address 08A48) and runs from there. If you use this in a multi-statement command or program line to exercise your target code, you can make the debugger start emulating your code without keyboard direction. Using the example above, if you execute:

DEBUG * @CALL BINTEST

the debugger will take over and begin emulation of your binary file.

Register Contents. Executing **DEBUG** * will cause the previous values of the emulated registers to be lost. The registers take on values necessary for an exit through "NXTSTM".

Debugger Non-programmable

The **DEBUG** statement is programmable; but the debugger, except for a few specific actions, is not. Executing **DEBUG** in a program will suspend the HP-71 operating system and give keyboard control to the debugger, for manual direction. Executing **DEBUG** * acts similarly, but starts emulation automatically.

---

* All registers are restored except CPU status bits S15-S12. In addition, RAM contents may have been changed when working in the BASIC environment.

Exiting the Debugger

When you exit the debugger you may continue through your target assembly language, drop into the HP-71 warm start code where the actions of an "INIT: 1" are taken, or, in extreme cases, you can reset memory before returning to the BASIC environment.

There are four methods of manually exiting the debugger, each with different effects. Only two are described in detail here so the first-time user can feel comfortable with entering and exiting the debugger. Methods 1 and 2 leave the debugger in control of the keyboard, whereas methods 3 and 4 deactivate the debugger.

1. Pressing [Q][Q] (the [Q] key twice, within 1½ seconds) drops the debugger into the HP-71 main loop, where the full BASIC operating system is available. Debugger registers are preserved so that a debugging session may be resumed at later reentry.
2. Pressing [J][J] (the [J] key twice, within 1½ seconds) while the PC register is in the display performs a hard jump to the PC. This key sequence is described in section 7, "Back and Forth from BASIC."
3. The RECOVER: 2 sequence is used to deactivate the debugger and return complete control of the computer to the HP-71 operating system.
4. The RECOVER: 3 sequence performs a Memory Lost, clearing all of memory and deactivating the debugger. This key sequence is described in section 7, "Back and Forth from BASIC."

[Q][Q]

You will find that the [Q][Q] sequence is very useful for interrupting debugging sessions to perform BASIC functions — purge or edit files, print CATalogs, check and set TIME, RUN programs, change output devices, etc. Then, if you want to resume your debugging session where you left off, execute the **DEBUG** statement. This restores all emulated CPU registers to their values before you performed [Q][Q].*

When you drop into the BASIC environment with [Q][Q],† the debugger retains control of the keyboard, although this is transparent to the user. This condition (operating in BASIC with the debugger in control of the keyboard) is indicated by the ((•)) annunciator being lit. Other than the annunciator being on, the computer operates as a normal HP-71. (Since the debugger is not deactivated, however, observe the precautions listed above, under "Activating and Deactivating the Debugger.")

RECOVER: 2

The familiar "INIT:" message generated by the [ON][/] keystroke (pressing the [ON] and [/] keys simultaneously) is replaced by "RECOVER:" when the debugger is active. Selecting RECOVER level 2 deactivates the debugger, so that complete control of the computer is returned to the BASIC operating system.

---

* Except CPU status bits S15-S12 and RAM contents which might have been changed while in the BASIC environment.

† The effect on the HP-71 operating system is identical to performing an INIT: 1, including turning off USER mode and reconfiguring all memory devices.

The RECOVER sequence can be used any time the debugger is active, whether viewing registers or after dropping into the BASIC environment with [Q][Q].

Follow these steps to deactivate the debugger with RECOVER: 2 :

1. If you have not activated the debugger yet, enter the **DEBUG** statement and press [ENDLINE]. This will activate the debugger with the PC register displayed.
2. Press and release both the [ON] key and the [/] key at the same time. This will generate the "RECOVER: 1" prompt.
3. Select RECOVER: 2 by pressing the [2] key, then [ENDLINE]. This deactivates the debugger and returns control to the BASIC environment.

For more details on leaving the debugger, different ways to reenter, and other methods of exiting, refer to section 7, "Back and Forth from BASIC."

Errors in File Setup

The debugger LEX files will not prevent you from copying them into any location in memory, but the **DEBUG** statement will generate an error or warning if you have not copied them into "protected" RAM (i.e., IRAM). You can run the debugger if any of the DBGLEX files reside in MAIN memory, but the HP-71 will report warnings, since you should be aware that MAIN memory is much more volatile than IRAMs. In any case, the debugger will error if DBGLEX1A does not reside at an "xx008" address.

Executing the **DEBUG** statement will display an error or warning under the following conditions:

Errors and Warnings when executing **DEBUG**					
Filename	In IRAM	In MAIN	ROM, PROM	Missing	Result
DBGMAINA				X	ERR:Excess Chars
DBGLEX1A	X *1				OK.
DBGLEX1A	X *2				WRN:Verify Address
DBGLEX1A	X *3				ERR:Verify Address
DBGLEX1A		X *4			WRN:File DBGLEX1A in MAIN
DBGLEX1A		X *5			ERR:Verify Address
DBGLEX1A			X		ERR:Verify Address
DBGLEX1A				X	ERR:Missing Module
DBGLEX2A	X				OK.
DBGLEX2A		X			WRN:File DBGLEX2A in MAIN
DBGLEX2A			X		OK.
DBGLEX2A				X	ERR:Missing Module
DBGLEX3A	X				OK.
DBGLEX3A		X			WRN:File DBGLEX3A in MAIN
DBGLEX3A			X		OK.
DBGLEX3A				X	OK.

*1 : First file in IRAM.
*2 : Not first file in IRAM, but at address "xx008".
*3 : Not first file in IRAM and not at address "xx008".
*4 : At address "xx008" in MAIN.
*5 : Not at an "xx008" address.

The above errors and warnings reported by the DBGMAINA file show the prefix **DBG** to identify the source; for example,

DBG ERR:Missing Module

DBGLEX Files in MAIN

Running the debugger with any of the DBGLEX files in MAIN RAM is extremely risky. If you keep at it long enough, a Memory Lost is unavoidable, since the file chain in MAIN memory moves frequently.

Renaming the DBGLEX Files

Several copies of the DBGLEX files can reside in memory simultaneously. The debugger will choose which modules are in the most stable memory locations (in IRAM, or in the case of DBGLEX2A and DBGLEX3A, in IRAM/ROM/PROM) and use these in execution.

You can rename the files as you wish without affecting the debugger's operation. DBGLEX1A contains all the register information needed to restore a debugging session, including all options settings and HP-IL display control (but not including CPU status bits S15-S12 and RAM locations which might have been changed while in the BASIC environment). Hence, you can keep copies of it under different names for different sessions. If you rename the DBGLEX files, the above warnings will report their actual names. However, this manual refers to them as DBGLEX1A, DBGLEX2A, and DBGLEX3A.

When the **DEBUG** statement is executed, the controlling routines search memory for the DBGLEX files. The files are not found by name, but by internal coding which identifies their origin (whether DBGLEX1A, DBGLEX2A, or DBGLEX3A) and their version (VER$ response). Only two characteristics are required for the debugger to accept a DBGLEX file, regardless of its name:

1. It must reside in an appropriate device and address.
2. It must be the same VER$ designation as the DBGMAIN file.

Stacks and Registers

This section provides an overview of the debugger stacks and registers. Detailed information, as well as directions for editing, can be found in section 5, "Register Details."

Conceptually, the debugger's registers are arranged in stacks. There are six stacks of emulated CPU registers and four control stacks. Figure 1 shows the relative positions of stacks and registers.

Accessing the Stacks

Each of the ten stacks can be accessed with a "direct access" digit key, as indicated in Figure 1. In addition, the six CPU stacks make up a revolving set, accessible by scrolling left and right (with wrap-around) with the arrow keys. Within each stack, the registers are viewed by rolling the stack with the up- and down-arrows.

For example, when the debugger is first entered, the User's stack is empty and the PC register is displayed:

PC displayed:	PC:00000: 2034EE100060

Pressing [→] moves to the A-D stack, at the top level:

[→] to the A-D stack:	A: 0000000000000000 0
[↓] rolls the stack to B:	B: 0000000000000000 0
[↓] rolls the stack to C:	C: 0000000000000000 0
[↓] rolls the stack to D:	D: 0000000000000000 0
[↓] wrap-around to A:	A: 0000000000000000 0
[→] to the R0-R4 stack:	R0: 0000000000000000
[→] to the RSTK stack:	L0: 00000
[→] to the D0-IN stack:	D0:00000: 2034EE100060
[↑] wrap-around to OUT/IN:	OUT:000 IN:0000(0) I:E

Pressing [→] now wraps around the set of CPU stacks back to the User's stack. If the User's stack is empty, it is skipped and the PC stack is brought up:

[→] to the PC stack:	PC:00000: 2034EE100060

The control stacks can only be selected by the direct access keys. Pressing [7] brings up the Window stack:

[7] to select Window stack:	00000.2034EE100060F481
[8] to select Break Point stack:	0 BPs. RTN Level:NONE
[9] to select Options stack:	OPTIONS: pkoslcrbhevd

CPU Stacks

[0] *	[1]	[2]	[3]	[4]	[5]
User's	PC	A-D	R0-R4	RSTK	D0-IN
< >	PC	A	R0	L0	D0
< >	PCcy	B	R1	L1	D1
< >	XC/AC	C	R2	L2	ST
< >		D	R3	L3	HS
< >			R4	L4	OUT/IN
< >				L5	
< >				L6	
< >				L7	

[←] --------- scroll --------- [→]
through with
arrow keys

Control Stacks

[6]	[7]	[8]	[9]
Home	Window	BrkPts	Options
Home	W 1	#BPs	OPTIONS
	W 2	BP 1	TRC
	W 3	BP 2	BIAS
	W 4	BP 3	AUTO
	W 5	BP 4	XQT
	W 6	BP 5	
	W 7	BP 6	
	W 8	BP 7	
	W 9	BP 8	
	W10	BP 9	
	W11	BP10	
	W12	BP11	
	W13	BP12	
	W14	BP13	
	W15	BP14	
		BP15	

* direct access keys

PC: CPU Program Counter
PCcy: PC, Carry, Pointer, and Mode
XC: Last executed PC
AC: Action Completed

A-D: CPU arithmetic registers
R0-R4: CPU scratch registers
L0-L7: Levels 0-7 of CPU RSTK

D0-D1: CPU data pointers
ST: CPU software Status bits
HS: CPU Hardware Status bits
OUT/IN: OUT and IN registers

Home: Last register displayed

W1-W15: RAM/memory windows

#BPs: Number of Break Points
BP1-BP15: Break Point registers

OPTIONS: Control Options
TRC: Trace register
BIAS: Address Bias register
AUTO: Auto disassemble
XQT: Single opcode execute

Figure 1. Debugger Stacks and Registers

The direct access keys can also be used to select CPU stacks:

[1] to select PC stack:	PC:00000: 2034EE100060
[2] to select A-D stack:	A: 0000000000000000 0
[3] to select R0-R4 stack:	R0: 0000000000000000
[4] to select RSTK stack:	L0: 00000
[5] to select D0-IN stack:	D0:00000: 2034EE100060

The "Home" register is the last register displayed before running the emulator. It is useful for tracking the contents of a register between emulation steps. Upon first entering the debugger the Home register defaults to the PC register. When the User's stack is empty, selecting it with the direct access key [0] displays "< >".

[6] to select Home register:	PC:00000: 2034EE100060
[0] to select User's stack:	< >

Register Overview

The following is an overview of the debugger stacks and registers. More information on each register is provided in section 5, "Register Details."

User's Stack. The User's stack is an expandable set of registers, from zero to eight, which you can select. When the debugger is first entered, this stack is empty.

PC Stack. The PC stack is the principal stack for the debugger. It directs the emulator to the address for emulation, provides a one-step history of emulation, and shows important CPU control registers.

PC register:	PC:00000: 2034EE100060
PCcy register:	PC:00000 C:0 P:0 M:H
XC/AC.register:	XC:00000: -ATTN-

- The PC register shows the current PC (CPU Program Counter) for emulation and 12 nibbles of memory. (Setting the "D" option, in the OPTIONS register, with DBGLEX3A in memory will display the opcode mnemonic instead.)
- The PCcy register shows the current PC, the Carry, Pointer, and arithmetic Mode.
- The XC/AC register shows the address of the last executed PC and the Action Completed — the reason for interrupting the emulator. AC codes are listed in section 5, "Register Details."

A-D Stack. The four registers in the A-D stack emulate the arithmetic registers in the CPU: A, B, C, and D.

A register:	A: 0000000000000000 0
B register:	B: 0000000000000000 0
C register:	C: 0000000000000000 0

D register:	D: 0000000000000000 0

Since the P register (Pointer) greatly affects operations on the arithmetic registers, the value of P is indicated by separation within the digits of the registers. In the above displays, nibble 0 (the right-most digit) is separated from the rest, indicating that the P register has a value of 0. This is explained fully in section 5.

R0-R4 Stack. The five registers in the R0-R4 stack represent the emulated CPU scratch registers: R0, R1, R2, R3, and R4.

R0 register:	R0: 0000000000000000
R1 register:	R1: 0000000000000000
R2 register:	R2: 0000000000000000
R3 register:	R3: 0000000000000000
R4 register:	R4: 0000000000000000

RSTK Stack. The eight registers in the RSTK stack represent the emulated return levels: L0 through L7. The L0 register is the "top" return stack level — the next one to be popped off for a "RTN" or "C=RSTK" instruction.

L0 register:	L0: 00000
.	
.	
.	
L7 register:	L7: 00000

D0-IN Stack. The five registers in the D0-IN stack keep track of data pointers, status bits, and the OUT and IN registers.

D0 register:	D0:00000: 2034EE100060
D1 register:	D1:00000: 2034EE100060
ST register:	ST:0000 0000 0000 0000
HS register:	MP:0 SR:0 SB:0 XM:0
OUT/IN register:	OUT:000 IN:0000(0) I:E

- The D0 and D1 registers show the address of each data pointer and 12 nibbles from memory at that address.
- The ST register shows the sixteen CPU status bits. S15 is the left-most digit; S0 is the right-most digit.
- The HS register shows the four hardware status bits.
- The OUT/IN register shows the last value written to the OUT register, the user-specified contents of the IN register (as read by an "A=IN" or "C=IN" instruction), and the current status of the Interrupt register. (More information on these is provided in section 5.)

Home Register. The Home register is that register which was displayed at the last time the emulator was run. It is useful for tracking a register between steps of the emulator, when you might be looking at several different stacks.

Window Stack. The debugger provides 15 memory windows, W1 through W15. When the debugger is activated for the first time, they are all set to address 00000. Each window register shows 16 nibbles of memory for the address specified.

```
W 1 register:          00000.2034EE100060F481
   .
   .
   .
W15 register:          00000.2034EE100060F481
```

There are seven types of windows, described in section 5. The default type is "Direct", indicated with a "." symbol in the display.

Break Point Stack. The Break Point stack always contains at least one register — the #BPs register. Up to 15 break points can be added to this stack.

```
#BPs register:          0 BPs. RTN Level:NONE
BP 1 register:         BP 1~00000   (__   )Halt
   .
   .
   .
Next available BP:     BP 5~_____   (__   )Halt
```

The #BPs register indicates how many break point addresses are in the stack. The RTN Level indicator tells at which RSTK level the emulator will break: 0-7, ALL, or NONE.

For each Break Point register the break address and two special fields are shown. The field in parentheses is the Show register — the name of the register to be displayed if the break point is encountered. When not specified (indicated by "__", the default), the Show register becomes the top level of the User's stack, or, if empty, the PC register. The term "Halt" is the Break Point Action. You can choose between three actions: "Halt", "Cont", or "Jump". More information on these is in section 5, "Register Details."

Options Stack. This stack contains five registers to control the options and flow of the debugger.

```
OPTIONS register:      OPTIONS: pkoslcrbhevd
TRACE register:        TRC #00000   (__   )Halt
BIAS register:         BIAS:00000,00000-00000
AUTO register:         AUTO DSASSMBL @ 00000
XQT register:          XQT: 00
```

- The OPTIONS register shows twelve options available to control the emulator and debugger output.
- The TRACE register allows you to specify a register for continuous display during emulation, as well as the frequency of display. The first five digits are the count (in hex) between displays. The next two fields are the Trace Show and Trace Action fields, identical to the Break Point Show and Break Point Action fields described above.
- The BIAS register allows you to adjust displayed addresses, in a designated range, to a different base address. This is useful for following code on a printout where the assembler address does not match the machine address, as is often the case. The bias is applied to address registers only if the "B" option is set in the OPTIONS register. The BIAS option can

be active only if DBGLEX3A is present in memory.

- The AUTO register specifies an address which is used for automatic opcode disassemble (available only if DBGLEX3A is in memory). Pressing f[AUTO] causes a continuous stream of opcode mnemonics to be displayed, starting at the AUTO address. The stream is interrupted by pressing [ATTN].
- The XQT register is used to execute single opcodes through the emulator. Opcodes are entered as hex digits. Entering an opcode causes the emulator to be stepped once, as if the opcode were found at the current PC.

Section 4

Using the Keyboard

The debugger is a keystroke-driven processor overlaid upon the HP-71 operating system. To use the debugger to its full potential you should learn the stack structure (containing the emulated CPU registers) and the keyboard interface (for controlling the debugger/emulator).

Two factors complicate the keyboard interface: 1) the debugger shares the keyboard with the HP-71 operating system, and 2) the system offers many features, so that many of the keys are used to select special functions. You will find that these functions have been chosen to match the typing aids on the HP-71, so a keyboard overlay, which might interfere with a target application, is not necessary.

Figure 2 is a representation of the HP-71 keyboard, with the debugger's keys highlighted. The keys in bold outline are the only keys used in the debugger, and the typing aids shown are the ones used as mnemonics for corresponding debugger functions.

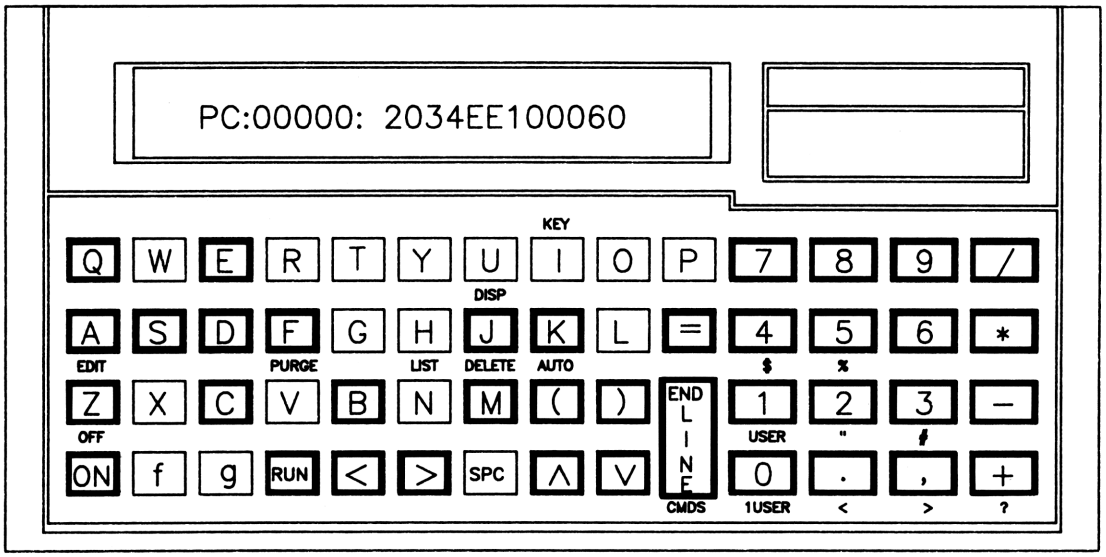


Figure 2. HP-71 Debugger Keyboard

The debugger operates in two modes: view mode and edit mode. In view mode the cursor is off, the arrow keys scroll through the stacks, and the digit keys are used as direct access keys to select stacks. In edit mode the cursor is on (blinking), the right- and left-arrow keys control the cursor, and the digit keys are used to edit register contents. In most cases the up- and down-arrow keys are inactive.

Key action with the debugger is similar to that of the HP-71 operating system. Keys repeat if held down for more than one-half second. *The most notable difference is that the debugger does not detect multiple keys*. For instance, in the BASIC environment you can press the [A] key, and while holding it down, press in turn [B], [C], [D], and [E]. All these characters will be detected and displayed. In the debugger, only one key at a time can be detected, including the f- and g-shift keys. *You must let up on the shift key before you press the second key in the keystroke*. You will become accustomed to this after a few debugging sessions.

As in the HP-71 operating system, the debugger allows type-ahead. If you press keys faster than the system can process them visibly, they are stored in the key buffer. Up to 15 keystrokes can be stored during type-ahead.

Key definitions made in the BASIC environment are inaccessible when directing the debugger, regardless of the User mode setting. The debugger detects and processes keys without intervention from the operating system.

Operating in View Mode

In view mode all keys are direct-execute. That is, they are executed immediately, with no need to press [ENDLINE].

Cursor Keys

The four cursor (arrow) keys, [←], [→], [↑], and [↓], are used to scroll through the set of six CPU stacks. As each stack is brought up, the top level is displayed. Figure 1 illustrates the stack structure.

Digit Keys

The ten digit keys, [0] through [9], are used as direct access keys to bring up the corresponding debugger stacks. Figure 1 shows the direct access key for each stack.

[Z]

Pressing [Z] (the key with the "EDIT" typing aid) enters edit mode. The next subsection, "Operating in Edit Mode," explains this mode.

[=]

Pressing [=] when viewing a *menu edit register* enters edit mode and allows selection of a menu address. The next subsection, "Operating in Edit Mode," describes the menu edit registers.

[(] and [)]

Pressing either [(] or [)] when viewing a Break Point or TRACE register enters edit mode and allows selection of a Show register or Action. The next subsection, "Operating in Edit Mode," explains edit mode and Show and Action editing.

f[EDIT]

Pressing f[EDIT] allows you to edit *memory contents* in an address register. The next subsection, "Operating in Edit Mode," explains memory editing.

f[DELETE]

Pressing f[DELETE] resets the register to the default contents. In almost all cases, this means to set all digits to zeroes. Special cases are as follows:

Mode. In the PCcy register, Mode is reset to show "M:H" (Hex).

RTN Level Indicator. Resets the RTN Level indicator in the #BPs register to "NONE".

Break Point Register. Deleting a Break Point register removes it from the Break Point stack, reducing the number of break points by one. The Show register is reset to "__" (null), which causes the User's stack (or, if empty, the PC register) to be displayed at a break. The Action field is reset to "Halt".

OPTIONS Register. Clears all options (sets all to lower case).

XQT Register. Resets the opcode to "00" without executing it.

Trace Show and Action. Resets the Show register to "__" (null), and the Action to "Halt".

f[PURGE]

Pressing f[PURGE] resets the entire current stack and brings up the top level in the stack as the new current register. Use this keystroke to "delete" all break points, or to set registers A through D to zeroes, for example. f[PURGE] is ignored in the User's stack.

Address Increment/Decrement

Several debugger registers are *address monitors* — that is, they, or the areas of memory to which they point, might require frequent changes to control the flow of a debugging session. These registers are:

- PC and PCcy registers (PC address)
- D0 and D1
- W1 through W15
- BP1 through BP15
- AUTO DSASSMBL register

Four keys can be used to increment or decrement the address in these registers without switching the debugger to edit mode:

[+] Increments address by 1.
[-] Decrements address by 1.
[*] Increments address by 10hex (16dec).
[/] Decrements address by 10hex (16dec).

[S]

Pressing [S] single-steps the emulator at the address in the PC register. Section 6 provides more details on running the emulator.

[M]

Pressing [M] macro-steps the emulator starting at the address in the PC register. A macro-step is identical to a single-step, except when a GOSUB (or GOSUBL or GOSBVL) instruction is executed. Then the emulator runs through the subroutine, breaking at the return (or any other break point if it occurs within the subroutine). A break will also occur if the flagged RSTK level is popped off with a "C=RSTK" instruction.

Section 6 provides more details on running the emulator.

[RUN]

Pressing [RUN] runs the emulator, starting at the address in the PC register. Emulation continues until a break is encountered, corresponding to one of the AC codes described under the heading "XC/AC register" in section 5, "Register Details." You can always break the emulator with the [ATTN] key; other breaks are caused by break points or special opcodes. Section 6 provides more details on running the emulator.

[Q][Q]

Pressing [Q][Q] (the [Q] key twice, within 1½ seconds) exits the debugger, back to the BASIC main loop, in effect executing an "INIT: 1". Refer to section 7, "Back and Forth from BASIC," for details.

[J][J]

Pressing [J][J] (the [J] key twice, within 1½ seconds) causes a hard jump to the address in the PC register. This keystroke can only be executed if the PC or PCcy register is in the display. Refer to section 7, "Back and Forth from BASIC," for details.

Memory Window Keys

Most debugger registers will display a memory window or opcode mnemonic from the address (or address field) they contain. These registers are:

- PC and PCcy
- XC
- The arithmetic (A-D) and scratch (R0-R4) registers, which frequently hold an address in the low five nibbles.
- L0 through L7
- D0 and D1
- W1 through W15
- BP1 through BP15
- AUTO DSASSMBL

Momentary Nibble Window. Several keys are used to display memory windows from the address registers. The window is displayed until the key is released.

[.] key. Pressing [.] momentarily displays a direct window — 16 to 19 nibbles read directly from the register address.

Example: Suppose a Break Point register is set as follows:

```
BP 1~09656  (__  )Halt
```

To see memory contents at the BP address,

press and hold [.]:

```
BP 1. 71801F4869011B8A
```

The "." in the display indicates a direct window, matching the character of the key pressed.

g[<] key. Pressing g[<] momentarily displays an indirect window. A five-nibble address is read from memory at the window and the memory contents at the new address are displayed.

Example: From the above example, you can see that the indirect address at 09656 is 10817. The indirect address is found by reading the first five nibbles (in reversed order, as performed in the CPU) from memory.

To see indirect memory contents from the BP address (the memory contents from address 10817),

press and hold g[<]:

```
BP 1< 9169028902AD9B2B
```

The "<" in the display indicates an indirect window, matching the character of the key pressed.

Momentary Disassemble Windows. Several keys are also used to display disassembled opcodes from the address registers. The mnemonic is displayed until the key is released.

The disassembler is available only if DBGLEX3A is in memory.

[,] key. Pressing [,] momentarily displays a direct disassemble — the opcode mnemonic read directly from the register address.

Example: Using the above example, you can display the opcode mnemonic at the address in the Break Point register, address 09656:

press and hold [,]:

```
BP 1. GOSUB 096DB
```

The "." in the display indicates a direct window disassemble.

g[>] key. Pressing g[>] momentarily displays an indirect disassemble — a five-nibble address is read from the memory at the window, and the mnemonic found at the new address is displayed.

Example: From the above example, you can see that the indirect address at 09656 is 10817. To display the opcode mnemonic from address 10817,

press and hold g[>]:

```
BP 1< ?C#A WP/GOYES 10823
```

(Only the last 22 characters are displayed in the LCD, so, in this case, the characters "BP " are not shown.)

The "<" in the display indicates an indirect window disassemble.

Relative Windows. There are five relative window keys:

- g[!] — to display a REL(1) window.
- g["] — to display a REL(2) window.
- g[#] — to display a REL(3) window.
- g[$] — to display a REL(4) window.
- g[%] — to display a REL(5) window.

Note that each relative window key corresponds to the digit on the primary key face.

Pressing any of the relative window keys momentarily displays the relative address computed from the window address. The display will continue for 1½ seconds after you release the key. During that time, you can press [.] or [,] for a nibble or disassemble window, respectively, on the relative address.

Example: Suppose you are viewing the PC register:

PC in display:	PC:01CC9: 47172F385885
g[#] for REL(3) window:	PC# [01CC9]=> 01E3D
and then, within 1½ seconds,	
[.] for memory window:	PC# 1BB74F215C06F2D739

The REL(3) offset is found by reading three nibbles from memory — in this case, "174". The REL(3) address from 01CC9 is computed by adding 01CC9+00174, giving the address 01E3D. At address 01E3D are found the nibbles 1BB74F215C06F2D739.

The "#" in the display indicates a REL(3) window, matching the character of the key pressed.

Relative memory windows are computed using a five-nibble sign extension in the same manner as the CPU; that is, if the high bit is set in the offset field, the entire offset is taken as negative.

f[USER]

To add a register to the User's stack, press f[USER] when the desired register is in the display. A beep will be sounded to signal that it has been pushed onto the User's stack. This does not make the User's stack the current stack; to do that, press the direct access key [0].

If the register is already in the User's stack, f[USER] will be ignored. A maximum of eight registers may reside in the User's stack. If a ninth register is pushed in, the last register in the stack is pushed off and lost.

Break point registers cannot be selected for the User's stack. However, the top level of the BP stack, the #BPs register, can be selected.

g[1USER]

To remove a register from the User's stack, select the User's stack and, using the up- and down-arrow keys, bring up the desired register in the display. Pressing g[1USER] removes it from the stack.

When all eight levels of the User's stack have been deleted, selecting the stack will show an empty

register: "< >".

f[LIST]

Pressing f[LIST] sends to the display all registers selected in the User's stack. The current register is redisplayed after this operation. f[LIST] is most useful for sending a quick listing of selected registers to the HP-IL display device.

[ENDLINE]

In view mode, pressing [ENDLINE] sends the display contents to the HP-IL display device, if it is active (i.e., if the "H" option is set). If the "E" option is also set, this is the only time registers are sent to the display device, except for a display generated by an emulator break.

[ATTN]

Pressing the [ATTN] key in view mode clears the key buffer and rebuilds the current register in the display. Pressing [ATTN] clears any type-ahead keys, avoiding action on them.

f[OFF]

Pressing f[OFF] causes the HP-71 to turn off, with the debugger still active. It also clears the debugger's key buffer, restores the LCD row data, and resets display contrast to the default value of "9".

The f[OFF] keystroke is most useful for restoring the display in the case that emulated routines have written to the display control area. The debugger can be turned back on with the [ATTN] key.

It is recommended that you not turn off your computer for any length of time when you are in the debugger, as it does not process wakeups as comprehensively as the mainframe does. To turn the computer off for any length of time, you should first drop into the BASIC environment using one of the four methods described in section 7, "Back and Forth from BASIC."

f[DISP]

Pressing f[DISP] momentarily displays the HP-71 emulated display buffer, which otherwise cannot be seen since the debugger overwrites the LCD. The display is maintained until you release the key. The display image is built in the LCD only and is not sent to the HP-IL device.

If the "L" option is set in the OPTIONS register, pressing f[DISP] will display the *LCD image* from the emulated routines. If this option is clear, pressing f[DISP] will display the *contents of the emulated display buffer*. In many instances, these will result in the same display. But frequently, when running in the HP-71 operating system, the LCD does not reflect the display buffer contents. In fact, between sending a new string to the display buffer and building it in the LCD, the two could not match, since the LCD still contains the old display.

There are two other situations where the difference between the LCD image and the display buffer contents is important: during scrolling and when WINDOW is in effect. If the emulated routines are using either of these features and you want to see the effect on the display, then set the "L" option in the OPTIONS register before using f[DISP].

f[AUTO]

Pressing f[AUTO] invokes the disassembler at the address specified in the AUTO DSASSMBL register. Opcode mnemonics are sent to the display continuously until [ATTN] is pressed. Opcodes are disassembled sequentially in memory, not by execution flow. The address in the AUTO DSASSMBL register is updated as each mnemonic is displayed.

There is no "DELAY" time in the debugger, so f[AUTO] is most useful with a display or printer. The disassembler is available only if DBGLEX3A is in memory.

g[CMDS]

When the file DBGLEX3A is in memory, you have access to a menu of extended commands to control special debugger actions. Pressing g[CMDS] in view mode gains access to nine commands which can be scrolled up and down with the arrow keys. When the desired command is in the display, pressing [ENDLINE] will execute it. The extended command menu can be aborted by pressing the [ATTN] key.

The nine extended commands are:

```
Clear Display
Restore Debugger IO
Reset HPIL
Wndw + DBGADDR$(1)
Wndw + DBGADDR$(2)
Wndw + DBGADDR$(3)
Initialize Debugger
Set Checksum
Verify Checksum
```

When in the extended command menu only four keys are active:

1. [ATTN] key: aborts the command menu and redisplays the current register.
2. [↓] key: rolls the menu to the next lower entry (with wrap-around to the top).
3. [↑] key: rolls the menu to the next higher entry (with wrap-around to the bottom).
4. [ENDLINE] key. Pressing [ENDLINE] executes the indicated command and returns the display to the current register.

The nine extended commands are explained fully in section 9, "Additional Features and Operating Details."

Operating in Edit Mode

As explained in the previous subsection, "Operating in View Mode," you can change the contents of registers without the need to edit them, with keystrokes such as f[DELETE], f[PURGE], and increment/decrement. But these methods are limited. For full control over the contents of debugger registers you must enter edit mode.

Every register except XC/AC is editable, although certain registers will accept only appropriate digits, such as 0 and 1 for status bits.

When in edit mode the debugger will respond only to edit keys. That is, stacks cannot be scrolled, nor can the emulator be run. The normal method for leaving edit mode is to press [ENDLINE], which enters the edited register contents and returns to view mode.

This section describes the general use of keys in edit mode. In section 5, "Register Details," you will find detailed descriptions of each register and specific editing actions. In particular, consult that section for details on:

- IN register editing
- RTN Level indicator
- Break Point Show register and Action
- Trace Show register and Action
- OPTIONS editing
- XQT register editing

Entering Edit Mode

Pressing [Z] (the key with the "EDIT" typing aid) switches to edit mode and allows you to edit the displayed register. When in edit mode the cursor blinks in the display, indicating that the digit or character at that position is ready to be changed.

For selected registers other keystrokes will also enter edit mode. These keystrokes are f[EDIT], [=], [(], and [)]. The affected registers are described in the following paragraphs.

Returning to View Mode

The following keystrokes will exit edit mode and return to view mode. Refer to the descriptions of these keys in the previous subsection, "Operating in View Mode."

[ENDLINE]	Enters the edited contents. The [ENDLINE] key also causes the register display to be sent to HP-IL if the "H" option is set.
[ATTN]	Pressing [ATTN] aborts the edit, returning the register to its previous contents. It also clears the debugger's key buffer.
f[OFF]	Performs the same action as [ATTN], and in addition turns off the computer and restores the display control RAM.
f[DELETE]	Resets the register to the default contents (zeroes, in most cases) or, with a Break Point register, removes it from the stack. f[DELETE] may be executed in either edit or view mode.
f[PURGE]	Resets the entire current stack. f[PURGE] may be executed in either edit or view mode.

[Z]

The [Z] key (the key with the "EDIT" typing aid) is used to enter edit mode for changing the contents of a debugger register. (Contents of *memory* can be edited with the f[EDIT] keystroke, described below.)

Digit Keys [0]-[9], [A]-[F]

In edit mode, the digit keys [0]-[9] and [A]-[F] enter hex digits to change the contents of the register or memory as applicable. In some registers such as ST, only [0] or [1] are allowed.

Cursor Keys

The two cursor keys, [←] and [→], are used to move the cursor to an editable digit or character. The two shifted cursor keys, g[←] and g[→], move the cursor to far-left and far-right, respectively.

> **Example:** Suppose the routine you are emulating requires the value of P to be 0, but it is currently set to "F" (15 dec). Edit the P register to set it to the desired value.

PCcy register in display:	PC:1364C C:0 P:F M:H
[Z] to edit (cursor on '1'):	PC:1364C C:0 P:F M:H
[→] 6 times (cursor on 'F'):	PC:1364C C:0 P:F M:H
[0] to change P value:	PC:1364C C:0 P:0 M:H
[ENDLINE] to enter:	PC:1364C C:0 P:0 M:H

[=]

The [=] key when viewing a *menu edit register* will enter edit mode and allow selection of a menu address. Menu edit registers are described under "Menu Editing Keys," below.

[(] and [)]

The [(] and [)] keys when viewing a Break Point or TRC register will enter edit mode and allow selection of a Show register or Action. These are described under the appropriate headings in section 5, "Register Details."

f[EDIT]

Several debugger registers are *address monitors* — that is, they, or the areas of memory to which they point, might require frequent changes to control the flow of a debugging session. These registers are:

- PC and PCcy registers (PC address)
- D0 and D1
- W1 through W15
- BP1 through BP15
- AUTO DSASSMBL register

The f[EDIT] key is used to edit *memory* at the address shown in an address monitor register. In the Window stack, even though a register might be an indirect or relative window, f[EDIT] always

works on the memory *at the address shown in the display*.

f[EDIT] may be pressed in view mode or in edit mode. Repeatedly pressing f[EDIT] allows you to pull up indirect addresses for memory editing. For example, at address 00005 of the mainframe you will find the nibbles: E1000. If the PC is set to address 00005, pressing f[EDIT] will allow you to edit memory at that address (although, in this case, 00005 is in ROM, so the contents cannot be changed). At address 0001E are the nibbles 47190; at address 09174 are the nibbles 0003A.

PC in display:	PC:00005: E100060F4812
f[EDIT] (cursor on 'E'):	00005:E100060F4812C136
f[EDIT] (cursor on '4'):	0001E:47190115071902AF
f[EDIT] (cursor on '0'):	09174:003A41CFBE2AF8EF
and so on.	

The debugger provides no protection for editing memory. Not only can you write to private files, system pointers, and even the debugger itself, but there is no check made on the type of memory device. Using f[EDIT] for memory in ROM or EPROM will do no harm (and do nothing), but using it in EEPROM may wipe out the device's memory.

Menu Editing Keys

Several registers allow speed editing by means of a stack menu. Using this, you can set the address in your register to the value in any other debugger register.

The menu edit registers are the following, with examples of their displays:

• PC register:	PC:00000=__
• D0 and D1 registers:	D0:00000=__
• W1 through W15:	W 1/00000=__ Type/D.
• BP1 through BP15:	BP 1~00000=__ (..)..
• AUTO DSASSMBL:	AUTO@00000=__

In these registers, menu editing is indicated by the "=__" symbol that follows the address. To use menu editing, press [=]. You may have already entered edit mode (with [Z], [=] or, in the case of a Break Point register, [(] or [)]), or you can press [=] directly from view mode. This moves the cursor over the "=" character, and gives you a window on the debugger register stacks. Then the arrow keys, along with the direct access keys, can be used to select any register in the stacks.

Example: Suppose you have just stepped the emulator through a few instructions and found that you need to set the D1 register to the value in the C register (A field). You can use menu editing to do this:

[5] to select D0-IN stack:	D0:2F43C: 000000000303
[↓] to D1 register:	D1:2F599: 000A3000A300
[=] for menu edit, cursor on '=':	D1:2F599=__

The null menu entry signals that the address is not being copied from any debugger register; the address is the last value found in the edit buffer.

Now the digit keys [0] through [9] act as direct access keys for the debugger stacks, and the arrow keys are used to scroll through the stack registers.

[2] to select A-D stack in menu: D1 : 0 4 0 0 9=A

The address in the A register (A field) has been copied into the D1 edit buffer.

[↓] to B register in menu: D1 : 0 0 0 0 0=B

The A field of the B register has been copied into the D1 edit buffer.

[↓] to C register in menu: D1 : 3A5 5 8=C

The A field of the C register has been copied into the D1 edit buffer.

[ENDLINE] to enter new D1 value
and return to view mode: D1 : 3A5 5 8 : 15AF2 1 5AF2 0 8

During menu editing the User's stack is ignored. Pressing the direct access key [0] instead shows the null menu entry. Any register in the User's stack can be found in its normal place in the debugger stacks.

When editing a Break Point register, the Show and Action fields, because of the limited display, both appear as ".." to indicate their presence:

[Z] to edit: BP 1~0 0 0 0 0=__ (. .) . .

They are restored when you return to view mode or when you edit either of the fields.

All registers can be selected in the edit menu, but some do not have valid addresses associated with them. The following registers, if called up in the edit menu, will show the address 00000:

- ST register
- HS register
- OUT/IN register
- #BPs register
- OPTIONS register
- TRC register
- BIAS register
- XQT register

You can select the Home register for a menu address by pressing [6]. The name of the Home register will appear and you can scroll throughout the stack structure from that point.

Section 5

Register Details

This section explains each register in detail, with information on special editing actions. General editing procedures are described in section 4, "Using the Keyboard."

Some operations on certain registers can only be performed when the optional file DBGLEX3A is in memory; these operations are identified in the descriptions which follow. Attempting to execute them with DBGLEX3A missing will display the message "Missing Module."

At the start of each register description is a list of actions available to you when viewing or editing the specified register.

The User's Stack

As described in section 4, the keystrokes f[USER] and g[1USER] are used to add and remove registers from the User's stack. This stack has two special purposes:

1. It makes for convenient access to the most frequently viewed registers.
2. It is the default display stack for several operations, such as encountering a break point during emulation.

The USER annunciator is turned on when the User's stack is in the display.* This will help you keep track of where you are in the stack structure.

Press the direct access key [0] to move to the User's stack. If you have placed more than one register in the stack, you can scroll through with the up- and down-arrow keys. If you select this stack with the [0] key and it is empty, the null register "< >" will be displayed.

PC Stack

Press the direct access key [1] to move to the PC stack. Each of the registers can be viewed by scrolling with the up- and down-arrow keys.

---

* You can select another annunciator if this conflicts with your application. Refer to section 9, "Additional Features and Operating Details."

PC Register

- [Z] to edit PC address
- [=] for menu editing
- f[EDIT] to edit memory contents
- Address increment/decrement: [+] [-] [*] [/]
- Memory window keys: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]

The PC specifies the address for emulation. Both the PC and PCcy registers display the same PC address. The PC address is updated continuously during emulation, between each step.

The PC register displays 12 nibbles from memory or the disassembled opcode at the PC address, according to the "D" option setting. The disassembler is available only if DBGLEX3A is in memory.

PCcy Register

- [Z] to edit PC, Carry, Pointer, and arithmetic Mode
- f[EDIT] to edit memory contents
- PC address increment/decrement: [+] [-] [*] [/]
- Memory window keys: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]

Both the PC and PCcy registers display the same PC address. The PCcy register does not use menu editing. The Carry register accepts only digits "0" or "1". The Pointer register is displayed as a single hex digit, "0" through "F". The Mode is displayed as, and accepts in editing, only "D" for decimal and "H" for hex.

The P value can also be determined when viewing an arithmetic register, A through D, by separation between nibbles.

XC/AC Register

- Not editable
- Memory window keys: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]

This shows the address of the last executed PC, and the Action Completed — the reason for interrupting the emulator. Using the memory window keys, you can determine the opcode or mnemonic of the last instruction. The XC address is copied from the PC before the PC is updated during emulation.

AC Codes. The Action Completed codes displayed in the XC/AC register are as follows:

-ATTN-	The emulator was stopped by pressing the [ATTN] key. This is also the AC code when first entering the debugger.
S STEP	The emulator was single-stepped through an opcode, either by pressing the [S] key or the [M] key.
TRACE	The emulator was interrupted by a Trace action — either Trace Halt, which stops the emulator, or Trace Cont, which displays registers and continues emulation.
M STEP	The emulator was macro-stepped through a subroutine with the [M] key. A break occurred because the RSTK level was popped off by a "RTN" instruction or by a "C=RSTK".

BRK PT	The emulator was interrupted by a break point action — either Break Point Halt, which stops the emulator, or Break Point Cont, which displays registers and continues emulation.
SHUTDN	The emulator encountered a "SHUTDN" instruction (opcode 807), along with option "S" being set.
X RSTK	The emulator's RSTK stack was exceeded; that is, a return level flagged for a break was pushed off the end of the RSTK. A return level can be flagged for a break through macro-stepping (which normally breaks when the subroutine returns) or with the RTN Level indicator in the Break Point stack.
RTN BK	The emulator encountered a return level break. A return level break is set with the RTN Level indicator in the Break Point stack.
IN PROMPT	An "A=IN" or "C=IN" instruction was encountered (opcodes 802 or 803), along with the "?Prompt" setting in the IN register.
UNCNFG	The emulator encountered an "UNCNFG" instruction (opcode 804) which would unconfigure a device containing any of the DBGLEX files. The instruction cannot be emulated.
RESET	The emulator encountered a "RESET" instruction (opcode 80A). The instruction cannot be emulated. (Refer to the description of the "C" option in the OPTIONS register for a way to handle the most frequent occurrence of the RESET instruction.)
SHUTDN/PC=0	The emulator encountered a "SHUTDN" instruction (opcode 807) with the emulated OUT register having bit 0 cleared. Further emulation may result in a Memory Lost.
REENTER	The debugger was reentered from BASIC or assembly language, such as with **DEBUG** *, using the REENTR entry point, or from skimming the configuration routine (refer to the "C" option description). (The "P" option must be set in the OPTIONS register to obtain a Trace display with AC code of "REENTER".)
INITLZ	The debugger was initialized, or reset, with the "Initialize Debugger" command in the extended command menu.

A-D Stack

Press the direct access key [2] to move to the A-D stack. Each of the registers can be viewed by scrolling with the up- and down-arrow keys.

The four registers in this stack are identical in their operation.

- [Z] to edit contents
- Memory window keys, for A field address: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]

The value of the Pointer is reflected in these registers by separation of the digit to which it points. For instance, a P value of 3 causes the registers to show nibble 3 separated from the rest:

With P=3:	A: 6D031F9003E4 C 125
With P=B (=11dec):	A: 6D03 1 F9003E4C125
With P=F (=15dec):	A: 6 D031F9003E4C125
With P=0:	A: 6D031F9003E4C12 5

If you want to edit the C register, you can also use the XQT register to perform an "LCHEX" instruction.

R0-R4 Stack

Press the direct access key [3] to move to the R0-R4 stack. Each of the registers can be viewed by scrolling with the up- and down-arrow keys.

The five registers are identical in their operation.

- [Z] to edit contents
- Memory window keys, for A field address: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]

Note that the low five nibbles of R4 are unusable in assembly language, reserved for use in the interrupt routine. Since the debugger does not perform emulated interrupts, these nibbles do not change unpredictably as they do in real-time execution. The low five nibbles of the R4 register cannot be depended upon to represent the true contents of the real CPU.

RSTK Stack

Press the direct access key [4] to move to the RSTK stack. Each of the registers can be viewed by scrolling with the up- and down-arrow keys.

The eight registers L0 through L7 are identical in their operation.

- [Z] to edit contents
- Memory window keys: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]

If you want to push and pop values off the debugger's RSTK, you can execute opcodes "06" ("RSTK=C") and "07" ("C=RSTK") in the XQT register. Executing "GOSUB" and "RTN"-type instructions with XQT will also do this, but in addition these change the PC.

You can set a one-time break at a return level, or specify a break at all returns, with the RTN Level indicator in the #BPs register. This will help you to monitor RSTK usage by making use of several features:

— The emulator will break (AC code "RTN BK") when the indicated level is popped off by a "RTN" or "C=RSTK" instruction.

— You can monitor the shifting of the RSTK by tracing the RTN Level indicator.

— The emulator will break (AC code "X RSTK") if the indicated level is pushed off the RSTK by a "GOSUB" or "RSTK=C".

D0-IN Stack

Press the direct access key [5] to move to the D0-IN stack. Each of the registers can be viewed by scrolling with the up- and down-arrow keys.

D0 and D1 Registers

These two registers are identical in their operation.

- [Z] to edit address
- [=] for menu editing
- f[EDIT] to edit memory contents
- Address increment/decrement: [+] [-] [*] [/]
- Memory window keys: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]

You can set a break point in the Break Point stack so that a data read or write with DAT0 or DAT1 will halt the emulator. If a read or write occurs at (or spans) a break point address, this will cause a break.

ST Register

- [Z] to edit status bits

The ST register shows the sixteen CPU software status bits, in groups of four for readability. They are ordered the same as in standard CPU representation: S15 is the left-most digit, S0 is the right-most digit. Only digit keys [0] and [1] are accepted during editing.

Shared Status Bits. The debugger does not emulate the upper four status bits, S15-S12. Since these bits cannot be manipulated together as a register, the debugger allows direct access to them rather than emulating them in RAM. In the HP-71 BASIC operating system, these status bits are used almost exclusively for the following purposes:

— S15, when set, indicates TRACE mode is in effect within a BASIC program.

— S14, when set, indicates "Halt BASIC program."

— S13, when set, indicates "Program running."

— S12, when set, indicates "An exception has occurred." An exception is an event which may cause special processing by the BASIC main loop, such as the [ATTN] key being hit, or an HP-IL interrupt.

The debugger uses S12 to indicate "the [ATTN] key was pressed." Thus, no conflict arises because of the sharing of status bits S15-S12. However, if you drop into the BASIC environment and subsequently return to the debugger with the **DEBUG** statement, the previous values of S15-S12 might have been lost.

HS Register

- [Z] to edit hardware status bits

The hardware status bits are identified by name, and their relative positions are the same as in standard CPU representation:

```
MP:0   SR:0   SB:0   XM:0
```

Where

— MP is the Module Pulled bit
— SR is the Service Request bit
— SB is the Sticky Bit
— XM is the External Module Missing bit

Only digit keys [0] and [1] are accepted during editing.

OUT/IN Register

- [Z] to edit contents

This register shows three emulated read-only or write-only CPU registers. Although the OUT register is not truly readable, the debugger maintains a record of the last value from an emulated "OUT=C" or "OUT=CS".

Similarly, the IN register is read-only, but the debugger allows you to specify what value to read for an "A=IN" or "C=IN" instruction.

The Interrupt register is for information only, and reflects the last "INTON" ("E" for "Enabled") or "INTOFF" ("D" for "Disabled") instruction. It is not editable, and has no real effect on the emulator.

Editing the IN Register. There are four ways to specify the contents of the IN register. They are selected by pressing certain keys when the cursor is on the first digit of the IN register (the fourth digit in the display).

Digits. This is the default method. When first entering the debugger, the OUT/IN register reads:

```
OUT:000  IN:0000(0)  I:E
```

which specifies that "A=IN" or "C=IN" will read all zeroes. The digits method is selected by pressing any digit key ([0]-[9] or [A]-[F]) when the cursor is on the first digit of the IN register.

The digit in parentheses indicates a mask for the OUT register bits. For instance,

```
OUT:000  IN:0800(4)  I:E
```

Bit 2 (the 4's bit) of the OUT register corresponds to the second key row from the top ([A] through [*]). Thus, if the emulated OUT register has this bit set, a "C=IN" instruction will read "0800" — the [5] key — in this example. If bit 2 of the OUT register is not set, zeroes will be read.

Other examples of using the IN register:

To read only:	Set the IN register to:
[0] or [1] keys:	`OUT:00F  IN:0400(3)  I:E`
[W] or [T] keys:	`OUT:00F  IN:0012(8)  I:E`
Digit keys [1] through [9]:	`OUT:00F  IN:1C00(E)  I:E`

This method of specifying the IN register allows the most flexibility, for you are not limited to one key, and you have available all input lines.

KEY#. You can specify one key for the input register. This will cause an "A=IN" or "C=IN" instruction to read the appropriate input value only when the right keyboard row is energized in the output register.

To select this method of specifying the IN register, press f[KEY] when the cursor is in the first IN position. The display will prompt:

```
OUT:000  IN:KEY#__   I:E
```

and you can fill in the key number. Key numbers are found in the *HP-71 Owner's Manual*, page 123. They must be entered in decimal form, from 00 to 56. Key number 00 is a special code for the debugger which means "IN register=0000". The f- and g-shift keys are allowable, with key numbers 44 and 45, respectively. Key numbers greater than 56, or containing hex digits, are rejected; an error beep is issued and the debugger waits for proper input.

Note that key numbers are a subset of the digits method of specifying the IN register. For example, the following two displays are equivalent and specify the [5] key:

```
OUT:000  IN:KEY#26   I:E
OUT:000  IN:0800(4)  I:E
```

KEYBD. You can have the emulator read the actual HP-71 keyboard when it encounters an "A=IN" or "C=IN" instruction when running. This will cause the low four bits of the emulated OUT register to be copied to the real OUT register (regardless of the "O" option setting). Then the true IN register is read from the keyboard into A or C.

Select this method of specifying the IN register by pressing [K] when the cursor is in the first IN position. The display shows:

```
OUT:000  IN:KEYBD    I:E
```

When single-stepping (or macro-stepping) an "A=IN" or "C=IN" instruction, this option will cause a prompt, for certainly the [S] (or [M]) key will still be down. The next item explains an IN prompt.

IN PROMPT. You can have the emulator prompt for an IN register value when an "A=IN" or "C=IN" instruction is encountered. Select this option by pressing g[?] when the cursor is in the first IN position. The display will show:

```
OUT:000  IN:?Prompt  I:E
```

When the emulator encounters an "A=IN" or "C=IN", this will cause an "IN PROMPT" break. The IN PROMPT register will be displayed, already in edit mode:

```
OUT:001  <A=IN>:____ ?
```

or,

```
OUT:001  <C=IN>:____ ?
```

This display is unlike any other in the debugger, because it indicates the emulator has paused for input — as shown by the question mark. This is not a time to edit other registers, including the OUT register. After entering a value for the A or C register, pressing [ENDLINE] will resume emulation automatically.

The OUT value shown is the current value of the emulated OUT register. This would be important in order for you to determine what value to supply for the A or C register.

During an "IN PROMPT" only three keys are active besides the digit and cursor keys:

[ENDLINE]	Enters the desired IN register value and resumes emulation. If not all four spaces have been filled, the debugger prompts again.
[ATTN]	Aborts the edit and returns to view mode, with OUT/IN the current register. You can view and edit other stacks, if desired. Resuming the emulator (with [RUN], [S], or [M]) will pick up at the "A=IN" or "C=IN" instruction and reprompt.
f[OFF]	As [ATTN], plus it turns off the debugger.

Filling in the digits for an IN PROMPT does not assign a value to the IN register; it merely provides a one-time value for this "A=IN" or "C=IN" instruction.

Home Register

The Home register is reset upon pressing [S], [M], or [RUN]. After any emulator action (single-step, macro-step, or run) you may move around the stack structure to view different registers. You can always return to the last register displayed before emulation by recalling the Home register.

The direct access key [6] recalls the Home register.

Window Stack

Press the direct access key [7] to move to the Window stack. Each of the registers can be viewed by scrolling with the up- and down-arrow keys.

All fifteen registers are identical in their operation.

- [Z] to edit window address
- [=] for menu editing
- f[EDIT] to edit memory contents
- Address increment/decrement: [+] [-] [*] [/]
- Memory window keys: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]
- [/] to edit window-type, in edit mode

Because of the limited display, the register number is not shown. To obtain the register number, press a window key (such as [.]) or go into edit mode with [Z].

> **Example:** Suppose you have a window register in the display, and you want to move down in the stack to W 7. To find the window number of the current display, do the following:
>
> Window register in the display: 00005.E100060F4812C136

[.] for momentary direct window:	W 2. E100060F4812C1361
Or [Z] to enter edit mode:	W 2/00005=__ Type/D.

There are seven types of windows. Each displays 16 nibbles from the indicated address.

Direct window — shown with '.'. The address of memory is the window address itself.

Indirect window — shown with '<'. Reads the address (five nibbles in reversed order, as performed in the CPU) from the the window memory.

REL(1) window — shown with '!'. Reads one nibble from the window and uses it as an offset to compute a new address.

REL(2) window — shown with '"'. Reads a two-nibble offset from the window and uses it to compute a new address.

REL(3) window — shown with '#'. Reads a three-nibble offset from the window and uses it to compute a new address.

REL(4) window — shown with '$'. Reads a four-nibble offset from the window and uses it to compute a new address.

REL(5) window — shown with '%'. Reads a five-nibble offset from the window and uses it to compute a new address.

To select the window type, enter edit mode with [Z]. Press [/], which moves the cursor to the "Type/" position, on the "/" character. Now the up- and down-arrow keys are used to roll through the window-type menu.

Example: Suppose you are testing a routine which writes to the start of the operating system's available memory. The RAM location AVMEMS (address 2F594) stores a five-nibble pointer to the start of available memory. If you want to monitor memory contents at that location, use an indirect window:*

Window register in display:	00000.2034EE100060F481
[Z] to enter edit mode:	W 7/00000=__ Type/D.
Digits to edit AVMEMS address:	W 7/2F594=__ Type/D.
[/] to edit window-type, cursor on '/':	W 7/2F594=__ Type/D.
[↓] to roll window-type menu:	W 7/2F594=__ Type/I<
[ENDLINE] to enter:	2F594<45574D0F00442474

The contents of memory at available-memory-start are 45574D0F00442474. Now while you emulate your routine you can view the contents of memory without having to determine the address of available-memory-start.

To determine the *address* of available-memory-start, press [.]. This displays a direct window from which you can read the five-nibble indirect address.

---

* The memory displays are for example only. The contents of RAM in your HP-71 are certain to be different.

Window register in display:	2F594<45574D0F00442474
[.] for direct window:	2F594.5D873DEF93DEF93D

You can see that the address stored in AVMEMS is "378D5". This is the address of available-memory-start.

Example: A routine you are emulating uses a table of REL(3) offsets for computed GOTOs. Your table resides at address 312EA, and you want to determine the memory contents at the jump addresses.

Window register in display:	00000.2034EE100060F481
[Z] to enter edit mode:	W 1/00000=__ Type/D.
Digits to edit table address:	W 1/312EA=__ Type/D.
[ENDLINE] to enter:	312EA.11507902AF41507A

This is a direct window on the REL(3) table. The first three-nibble entry in the table is "511". When used as a three-nibble relative offset, the REL(3) address is 312EA + 00511 = 317FB.

[Z] to reenter edit mode:	W 1/312EA=__ Type/D.
[/] to edit window-type, cursor on '/':	W 1/312EA=__ Type/D.
[↑] to roll window-type menu:	W 1/312EA=__ Type/R%
[↑] to roll window-type menu:	W 1/312EA=__ Type/R$
[↑] to roll window-type menu:	W 1/312EA=__ Type/R#
[ENDLINE] to enter:	312EA#8502B70E3F41507A

The relative window is indicated by 'R' when editing the window type, and the accompanying symbol corresponds to the g-shifted digit key — in this example, the symbol 'R#' indicates a REL(3) window. The display shows that address 317FB contains the nibbles 8502B70E3F41507A.

The second three-nibble entry in the table, at address 312ED, was "970" (this can be seen from the direct window display above). The REL(3) address using this offset is 312ED + FF970 = 30C5D. (A five-nibble sign extension is performed when the high bit of the offset is set.)

Press [+] three times to increment the window address:	312ED#812F9624007314F7

The display shows that address 30C5D contains the nibbles 812F9624007314F7.

Window registers can be used for memory edit with the f[EDIT] keystroke. Even though a window register might be an indirect or relative window, f[EDIT] always works on the memory at the *address shown in the display*. To edit memory at the indirect address, press f[EDIT] again. Repeatedly pressing f[EDIT] will pull up successive indirect addresses for memory editing. Memory at a relative address cannot be edited with f[EDIT].

Similarly, using the momentary disassemble key [,] displays the opcode mnemonic at the *address shown in the display*. That is, [,] always displays the mnemonic at the direct memory only, even for

a Window register which is set for indirect or relative type. For an indirect disassemble, press g[>].

Break Point Stack

Press the direct access key [8] to move to the Break Point stack. Each of the registers can be viewed by scrolling with the up- and down-arrow keys.

#BPs Register

- [Z] to edit RTN Level indicator

The #BPs register shows how many break point addresses have been entered in the stack, from 0 to 15. The RTN Level indicator is editable, and provides a way to emulate the remainder of a subroutine and to monitor RSTK usage.

The allowable selections for the RTN Level indicator are 0 through 7, "ALL", and "NONE". The default is "NONE". To edit the indicator, press [Z] to enter edit mode. For 0 through 7, press the appropriate digit key. For "ALL", press [A]; for "NONE", press [N].

If a RTN Level break is set for 0 through 7, the level indicator will track the level number as the RSTK is pushed and popped. Any action which pops that level off the stack (any type of "RTN", or a "C=RSTK" instruction) will cause a "RTN BK". If the level is pushed off the end of the stack (any type of "GOSUB", or a "RSTK=C" instruction), an "X RSTK" break will occur. This is useful to insure that routines do not use too many return levels. After either a "RTN BK" or an "X RSTK" break, the RTN Level indicator is automatically reset to "NONE".

If you set the RTN Level indicator for "ALL", any type of "RTN" will cause a "RTN BK". However, a "C=RSTK" will not cause a break, nor will any type of RSTK push cause an "X RSTK" break.

Break Point Registers

All fifteen registers are identical in their operation.

- [Z] to edit Break Point address
- [=] for menu editing
- f[EDIT] to edit memory contents
- Address increment/decrement: [+] [-] [*] [/]
- Memory window keys: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]
- [(] to edit Break Point Show field
- [)] to edit Break Point Action field

Registers BP1 through BP15 (the break point addresses) can be entered and deleted as desired. A typical break point register looks like this:

```
BP 3~00209   (D1   )Halt
```

The address "00209" tells the emulator to break if the PC executes an instruction at this address (whether the opcode begins at this address, or if it spans this address). Also, the emulator will break if a data operation (reading or writing to memory with DAT0 or DAT1) occurs at this

address (whether the read/write begins at or spans this address).

If the emulator encounters a break point on the first nibble of an instruction, it will break *before* executing it. If a break is set on any other nibble of an opcode, it will first execute and then break. For a data operation (read/write with DAT0 or DAT1) the break always occurs before the instruction is executed. When single-stepping (or macro-stepping at a non-GOSUB instruction), all break points are ignored.

If the break point stack is not full (less than 15 break points entered), the last level appears as a prompt. For instance, if there are already three Break Points in the stack, rolling down to level 4 will show:

```
BP 4~_____  (__  )Ha l t
```

indicating that break point #4 is next to enter.

Break point registers are sorted in increasing order as you add, edit, or delete them. As you edit a break point, you may notice its register number change as it takes a new place in the stack. Ordering the break points greatly enhances the speed of the emulator.

Setting the "K" option in the OPTIONS register will cause a beep when a break point is encountered, as well as any other break except "-ATTN-", "S STEP", or "TRACE".

The break point registers allow you to specify a Show register (a register to display) and an Action to take when the break point is encountered by the emulator.

Editing the Show Field. Press [(] to edit the Show field. You may have already entered edit mode (with [Z], [=], or [)]), or you can press [(] directly from view mode. This places the cursor on the "(" and you now have a window on all the debugger stacks to select a Show register. Use the direct access digit keys and the arrow keys to select a register.

> **Example:** You want to change a Break Point address to 3AB19, and direct the emulator to display the ST register when it halts there.
>
> | | |
> |---|---|
> | Desired break point in display: | `BP 1~3A41C (__ )Ha l t` |
> | [Z] to enter edit mode: | `BP 1~3A41C=__ ( . . ) . .` |
> | Digits to edit BP address: | `BP 1~3AB19=__ ( . . ) . .` |
>
> When editing a break point, the Show and Action fields, because of the limited display, both appear as ".." to indicate their presence.
>
> | | |
> |---|---|
> | [(] to edit Show, cursor on '(': | `BP 1~3AB19 (__ )Ha l t` |
>
> The Show and Action fields are restored. Now while editing the Show register, you have a window on the debugger stacks. The arrow keys and the direct access keys are active to scroll through the stacks and bring up the name of any register.
>
> | | |
> |---|---|
> | [5] select D0-IN stack for Show: | `BP 1~3AB19 (D0 )Ha l t` |
> | [↓] roll down to D1 register: | `BP 1~3AB19 (D1 )Ha l t` |
> | [↓] roll down to ST register: | `BP 1~3AB19 (ST )Ha l t` |

[ENDLINE] to enter: BP 1~3AB19 (ST)Halt

Now when running the emulator, if this break point is encountered, the ST register will be brought up to the display immediately.

During Show editing the User's stack is ignored. Pressing the direct access key [0] instead shows the null entry: "(__)". (Any register in the User's stack can be found in its normal place in the debugger stacks.) A null Show entry indicates that, when encountering this break point, the debugger will display the top level of the User's stack by default. If the User's stack is empty the PC register will be displayed.

During Show editing, the Home register is not accessible by its contents — it is referred to by its formal name, "Home". Selecting "Home" as the Show register means that the debugger will display the Home register (the last displayed before running the emulator) at the break point.

A Break Point register can be selected for the Show register. As break points are reordered or deleted, the Show register number will change accordingly.

If you want the PC address displayed along with the Show register, set the "P" option in the OPTIONS register.

Special Show Register: XQT. Selecting the XQT register for Show causes a special action: instead of *displaying* the XQT register, the debugger also *executes* it. This option can be useful for "programming" the debugger. If you find, say, that a "P=0" instruction is missing from your routine, you can put the "20" opcode in the XQT register, set "XQT" and "Cont" for the BP Show and Action, respectively, and the missing instruction will be executed at the break point. If several instructions are missing, you can set the XQT register to perform a "GOSBVL" to some unused area of RAM where you have filled in the opcodes, or to an existing subroutine in the operating system.

Editing the Action Field. Press [)] to edit the Action field. You may have already entered edit mode (with [Z], [=], or [(]), or you can press [)] directly from view mode. This places the cursor on the ")" and allows you to select a Break Point Action.

There are three Break Point Actions available:

1. Halt — display Show register and stop emulator.
2. Cont — display Show register and continue.
3. Jump — display Show register and perform hard jump to PC.

Example: You want to trace the ST register through the instruction at address 3AB19; that is, you want to see the status bit settings every time the instruction is executed.

Desired break point in display: BP 1~3AB19 (ST)Halt

[)] to edit Action, cursor on ')': BP 1~3AB19 (ST)Halt

While editing the Action field, you have a window on the Action-type stack; the up- and down-arrows are used to roll through this stack and make a selection:

[↓] to roll Action-type menu: BP 1~3AB19 (ST)Cont

[ENDLINE] to enter: BP 1~3AB19 (ST)Cont

Now when running the emulator, if this break point is encountered, the ST register will be

brought up to the display immediately and the emulator will continue. This method can be used to trace any register, including tracking the RTN Level indicator to monitor the number of stack levels used.

The "Jump" option returns to the HP-71 BASIC operating system, restoring all real CPU registers from their emulated counterparts. This would be useful when a critical timing loop needs to be run at real speed, for instance. For information regarding hard jumps to the PC and reentering the debugger, refer to section 7, "Back and Forth from BASIC."

Options Stack

Press the direct access key [9] to move to the Options stack. Each of the registers can be viewed by scrolling with the up- and down-arrow keys.

OPTIONS Register

- [Z] to edit options

To set or clear options, press [Z] to enter edit mode. Now the up- and down-arrow keys are used to change the option letters to upper case or lower case, respectively. The arrow keys operate on the letter at which the cursor is blinking. An upper case letter indicates the option is set, while lower case indicates it is clear. When the debugger is first entered, all options are clear:

```
OPTIONS: pkoslcrbhevd
```

Some of the options have side effects on the running of the emulator. Refer to section 6, "The Emulator," for details.

P Option causes the PC address to be displayed along with the Show register for a Break Point or Trace.

The PC address is an abbreviated display of the PC register, which does not include the memory contents. For example, if the Trace Show register is ST, and "P" is set, the trace display might show:

```
PC:3AB19:
ST:0110 0101 0100 0000
```

Since the debugger does not have a "DELAY" time this is most useful with an HP-IL display device.

K Option causes the emulator to beep when emulation is stopped for any reason other than AC codes of "-ATTN-", "S STEP", and "TRACE". (Note that "RESET", "UNCNFG", and "SHUTDN/PC=0" will always beep, regardless of the "K" option.)

O Option causes an "OUT=C" or "OUT=CS" (opcodes 800 and 801) to be suppressed during emulation. That is, the true OUT register will not be written to.

S Option causes the emulator to break at a "SHUTDN" instruction (opcode 807). With "S" clear, emulation will continue through a "SHUTDN". (Regardless of the "S" setting, encountering a "SHUTDN" with bit 0 of the OUT register set to zero will always cause an emulator break, with an AC code of "SHUTDN/PC=0".)

L Option causes the LCD to be restored to its emulated image when invoking the emulator. With "L" clear, the display will be used to show the starting PC address for a run.

The "L" option also affects the action of the f[DISP] keystroke. Refer to the description in section 4, "Using the Keyboard."

C Option causes the debugger to allow the mainframe configuration routines to execute properly if the "RESET" instruction at address 10233 is encountered by the emulator. This is done by momentarily jumping into the operating system at that address, and intercepting the configuration poll at address 10F0C to recover. This action is referred to as "skimming the configuration routine."

If the "C" option is not set, the configuration routines will cause a break at the "RESET" instruction at address 10233.

R Option causes the debugger to handle service requests when it is idle in view mode. (This is done by issuing a service request poll when the hardware status bit SR is set.) This option is necessary to use KEYBOARD IS with the debugger, and also to keep the clock system up to date.

B Option causes the address bias (in the BIAS register) to be applied to the address registers. This is useful for adjusting displayed addresses to match program listings. For a list of affected registers, refer to the description for the BIAS register, below. The "B" option can be active only if DBGLEX3A is in memory.

H Option causes debugger register displays to be sent to the HP-IL display device. You must have first specified a debugger display device with the DDISPLAY IS statement.

E Option causes the debugger to output its display to HP-IL only when you press [ENDLINE]. If this option is *not* set, every display will be sent to the display device as you view registers. Regardless of the "E" option setting, any register display generated by the emulator (such as after a single-step, Break Point, or Trace) is always sent to the display device. If the "H" option is not set, this option has no effect.

V Option causes the debugger to display all CPU registers in an 80-column format, suitable for any HP-IL 80-column video device. Details on this are in section 8, "Using HP-IL." If the "H" option is not set, this option has no effect. The "V" option can be active only if DBGLEX3A is in memory.

D Option causes the PC register to display the disassembled opcode at its address, rather than 12 nibbles of memory. The "D" option can be active only if DBGLEX3A is in memory.

TRC Register

- [Z] to edit Trace count
- [(] to edit Trace Show field
- [)] to edit Trace Action field

The appearance of the TRC register is similar to that of a Break Point register. However, the first five digits are a hex count of the number of emulated opcodes between displays. A value of 00000 means no trace is in effect (actually 16^12, essentially an infinite count). The Trace Show and Trace Action fields are identical to those of Break Point Show and Action described above.

> **Example:** Your routine enters a loop of 26 instructions, and you want to monitor the value of the D1 register at every loop. Set the TRACE register as follows (26dec=1Ahex) and run the emulator:

```
TRC #0001A   (D1   )Cont
```

If you want the PC address displayed along with the Show register, set the "P" option in the OPTIONS register.

BIAS Register

- [Z] to edit BIAS fields

There are three five-nibble fields in the BIAS register: the base address for adjustment, the lower limit, and the upper limit of the applicable range. Addresses within the lower and upper limits (inclusive) are adjusted for display relative to the base address. That is, let:

B= base address
L= lower limit,
U= upper limit, and
A= an address in a debugger register.

Then, if L <= A <= U , the displayed address will be computed as B + (A - L) .

The registers to which the bias is applied are the following. The bias is only applied at time of display, not within the register itself.

- PC and PCcy registers (PC address)
- XC register
- L0 through L7
- D0 and D1
- W1 through W15
- BP1 through BP15
- AUTO DSASSMBL register
- Addresses of GOTOs, GOSUBs, etc., displayed by the disassembler.

The BIAS is active only if the "B" option is set and the DBGLEX3A file is in memory.

Example: You are emulating routines in the HP-IL ROM. The HP-IL IDS, Volume II, publishes source listings for the module with a starting address of F0008. In your HP-71, the ADDR$("HPILROM") function shows the ROM resides at address 40008. To adjust displayed addresses within the HP-IL module to match the listings, set the "B" option and edit the BIAS register as follows:

```
BIAS:F0008,40008-47FFF
```

This specifies that all absolute addresses between 40008 and 47FFF, inclusive, would be adjusted (for display only) to start at address F0008. Thus, in the example, address 40008 would be displayed as F0008, address 412C0 as F12C0, etc. Addresses outside the range are not affected.

Example: A LEX file you assembled with the FORTH/Assembler ROM resides in memory at 2FE14, extending to 301A7. The listing begins at address 00000. To display addresses within the file to match the listing, set the "B" option and edit the BIAS register to show:

```
BIAS:00000,2FE14-301A7
```

The BIAS is applied only when displaying an address register, not when editing. When you enter edit mode, the absolute address is displayed and you must enter absolute addresses only. Upon pressing [ENDLINE], the address will be redisplayed adjusted for the BIAS (if it falls within the BIAS range).

AUTO DSASSMBL Register

- [Z] to edit AUTO address
- [=] for menu editing
- f[EDIT] to edit memory contents
- Address increment/decrement: [+] [-] [*] [/]
- Memory window keys: [.] g[<] [,] g[>] g[!] g["] g[#] g[$] g[%]

The AUTO DSASSMBL register specifies where the debugger is to start displaying opcode mnemonics when f[AUTO] is pressed. Using the keystroke in view mode starts a continuous display of mnemonics until the [ATTN] key is pressed. Opcodes are disassembled sequentially in memory, not by execution flow. The address in the AUTO DSASSMBL register is updated as each mnemonic is displayed, so it can be resumed after being interrupted with the [ATTN] key.

The debugger, of course, has no way of determining if nibbles in memory are opcodes, BASIC tokens, or tables, or if the disassembly is being started at a correct opcode boundary.

The disassembler is available only if DBGLEX3A is in memory.

Example: You are emulating a subroutine at address 0ECBB, and you want to set a break point at the RTN instruction. Although you don't know the address of the RTN, the AUTO DSASSMBL register can be used to find it. Set:

```
AUTO DSASSMBL @ 0ECBB
```

and press f[AUTO]. The debugger will display a list of addresses and mnemonics:

```
0ECBB@ B=0 W
0ECBE@ SB=0
0ECC1@ CSRB
0ECC4@ ?SB=0/GOYES 0ECCC
0ECC9@ B=B+A W
0ECCC@ A=A+A W
0ECCF@ ?C#0 W/GOYES 0ECBE
0ECD4@ A=B W
0ECD7@ C=B W
0ECDA@ RTNCC
```

At this point, press [ATTN] to stop the disassembler. The address 0ECDA is the address of the subroutine RTN.

XQT Register

- [Z] to edit XQT opcode

The XQT register provides a means to execute a single opcode in the emulator. To execute an opcode, press [Z] to enter edit mode. Type in the desired opcode; the debugger interprets each

nibble to determine the appropriate length of your opcode, and will expand or contract the available nibbles to match it. When the opcode is entered, press [ENDLINE], which does two things:

1. executes the opcode in the emulator, and
2. sets the contents of the XQT register.

Once the contents of the XQT register are set, you can reexecute an opcode merely by pressing [Z], then [ENDLINE]. If you view other registers and come back to the XQT register, it will display the last opcode which was entered and executed (or the default "00", if the debugger was just initialized).

Example: The entry conditions for a certain subroutine require that P=5 and D0=2E2F8, but these registers are not set properly. You set a break point at the subroutine, and after the emulator breaks you use the XQT register to set P and D0:

[Z] to edit XQT:	XQT : 0 0
Digits to enter P=5 opcode:	XQT : 2 5
[ENDLINE] to execute opcode:	XQT : 2 5

The P register is now set to 5.

[Z] to edit XQT:	XQT : 2 5
Digits to enter D0=2E2F8 opcode:	XQT : 1B8F2E2
[ENDLINE] to execute opcode:	XQT : 1B8F2E2

The D0 register is now set to 2E2F8. Now you can press [RUN] to resume emulation in the subroutine. Notice that when you typed in the opcode digits, the register expanded to match the length of the opcode.

If while editing the XQT register, you decide to abort the edit and avoid execution, press [ATTN]. This will return the XQT register to its previous contents with no opcode execution. Similarly, pressing f[DELETE] resets the XQT register to "00" without executing the opcode.

The XQT register is also executed automatically by the emulator when it is selected as the Show register for Break Point or Trace.

Effect on PC. Using the XQT register is like *inserting* an instruction at the PC address — the instruction is executed, but unless the opcode involves a jump or return, the PC is not updated. The following paragraphs describe the effect on the PC for each class of opcode.

Non-PC Related Instructions. The majority of opcodes are in this class. These are:

— Arithmetic, such as "B=C", "A=A+A", "CR2EX", "LCHEX", etc.
— Pointer and data pointer, such as "P=2", "D1=C", "A=DAT0", "D1=D1+5", etc.
— Control, such as "SETHEX", "OUT=C", "INTOFF", "C=RSTK", etc.

These instructions are executed and the PC is not changed.

Relative Jump Instructions. These are:

— GOC (with carry set); GONC (with carry clear)
— A test instruction, jumping when true
— GOTO and GOLONG

For this class of opcode, XQT causes the PC to be adjusted by the relative offset of the jump.

Thus, if the PC address is currently "3A041", and you edit and enter in the XQT register a "GOTO +10B relative" instruction:

XQT: 6A0 1

the PC will be incremented by +10B to 3A14C.

For a GOC or GONC which does not branch, or for a test which fails, the PC is not changed.

Relative Subroutine Instructions. These are:

— GOSUB
— GOSUBL

For this type of instruction, the current PC is pushed into L0 of the RSTK as the return address. Then the PC is adjusted by the relative offset, as for a GOTO. The entire subroutine is not executed,* but only the single GOSUB instruction.

GOVLNG and GOSBVL. For a GOVLNG, the PC is simply set to the address of the jump. For a GOSBVL, the current PC is pushed into L0 of the RSTK as the return address, and then the PC is set to the address of the jump.

Return Instructions. All types of returns (RTN, RTNC, RTNCC, RTNYES, etc.) are handled by popping level L0 off the RSTK and setting the PC to this value.

For a RTNC, RTNNC, or RTNYES which does not return, the RSTK is not affected nor is the PC changed.

Similar to S STEP. Any action which could be expected when running the emulator may happen with the XQT register. For example, executing a "RESET" instruction (opcode 80A) will cause a beep and display the XC/AC register. The only exception is for an IN PROMPT, caused by executing an "A=IN" or "C=IN" (802 or 803) instruction with the IN register set to "?Prompt". This will cause the XC/AC register to be displayed, rather than the usual IN PROMPT register.

Although the XQT register allows up to 16 digits, only one opcode at a time can be executed. All instructions are allowed, except that an "LC(15)" (17 nibbles long) or "LC(16)" (18 nibbles long) can only be edited out to the 16th nibble. To execute these two instructions in the XQT register as a Break Point or Trace Action, you can edit two nibbles in RAM (location RegXQ+ in DBGLEX1A) to extend the opcodes. Refer to appendix C, "Addresses of Entry Points," for details.

The AC code for XQT is "S STEP".

---

* Unless the XQT register is being executed as a Break Point or Trace Show register, and the Action=Cont.

Section 6

The Emulator

Once you have mastered the debugger's stacks and registers, you will find the emulator easy to use. It emulates assembly language by manipulating RAM buffers as the real CPU manipulates its registers. The real power of the debugger is in its viewing and editing capabilities when the emulator is suspended.

The emulator runs at about 1/75th real speed. Routines which take about five seconds to execute in the BASIC environment will take more than six minutes to emulate, assuming no break points are encountered. One of the most noticeable effects will be to make the display building process visible. For those applications which use the display routines, you may find this a good way to study the display procedures.

Emulating at the PC

Make sure the PC is set to the correct address for emulation. Once you have set up the emulated registers as needed (either by editing them, or through the RECOVER sequences described in section 7), the emulator can be invoked from view mode by three methods:

1. Single-step: [S]
2. Macro-step: [M]
3. Run: [RUN]

Return to Home Register

In each method, when the emulator breaks, the debugger will display the Home register — the register displayed before [S], [M], or [RUN] was pressed — unless the break is for one of the following reasons, shown by AC code in the XC/AC register:

BRK PT. If the emulator is running, encountering a break point will cause it to display the Show register you selected.

After the break, you can run the emulator through the break point location with [S], [M], or [RUN].

TRACE. If the emulator is running and the TRC register is nonzero, after the specified number of instructions it will break and display the Show register you selected.

After the break, you can run the emulator through the trace location with [S], [M], or [RUN].

IN PROMPT. Encountering an "A=IN" or "C=IN" instruction with the IN register set to "?Prompt" will cause a break with AC code "IN PROMPT". The IN PROMPT register will be displayed in edit mode. You need to supply a value for the IN register to resume emulation.

SHUTDN. If the "S" option is set and a "SHUTDN" instruction (opcode 807) is encountered, the emulator will break and display the XC/AC register, AC code "SHUTDN".

After the break, you can run the emulator through the SHUTDN location with [S], [M], or [RUN].

SHUTDN/PC=0. If a "SHUTDN" instruction is encountered and the emulated OUT register has bit 0 set to zero, the emulator will beep and break with the XC/AC register displayed, AC code "SHUTDN/PC=0". This break action will take place regardless of the "S" option setting.

In this case, you can run the emulator through the "SHUTDN" instruction with [S], [M], or [RUN]. But since the PC has been set to 00000, this will execute the cold start routine, clearing memory and inevitably causing a Memory Lost.

UNCNFG. If an "UNCNFG" instruction is encountered (opcode 804) and the addressed device contains any of the DBGLEX files, the emulator will beep and break with the XC/AC register displayed, AC code "UNCNFG". You cannot step or run the emulator through this instruction.

RESET. If a "RESET" instruction is encountered (opcode 80A), the emulator will beep and break with the XC/AC register displayed, AC code "RESET".

If the "RESET" encountered is at address 10233 (in the operating system's configuration routine), you can run or single-step at this point after setting option "C". This causes the debugger to "skim through" the configuration routine: it performs a hard jump to 10233 and automatically resumes emulation at address 10F0C by intercepting the configuration poll.

If the "RESET" encountered is at any other address in memory, you cannot step or run the emulator through the instruction.

Single-Stepping

Press [S] to single-step the emulator. Holding down the [S] key will cause repeated single-steps.

After each single-step, the PC is incremented to the next instruction (except for AC codes "IN PROMPT", "UNCNFG", and "RESET", described above). You can single-step through a break point or a "SHUTDN" instruction.

The AC code after a single-step is "S STEP", unless one of the above special breaks occurs. Setting the "K" option will help to identify them, since a beep will be sounded at any break other than "-ATTN-", "S STEP", or "TRACE".

Macro-Stepping

Press [M] to macro-step. This is identical to single-stepping in all cases except for GOSUBs (including GOSUBL and GOSBVL). For a GOSUB, the emulator will continue running until the subroutine returns or until another reason for break occurs within the subroutine. If a "C=RSTK" instruction pops the flagged level, a break will also occur.

For macro-step the AC code is "S STEP" unless a GOSUB is encountered. In this case, when the flagged RSTK level is popped off, the emulator will break with AC code "M STEP". Setting the "K" option will cause a beep at any break other than "-ATTN-", "S STEP", or "TRACE"; this will alert you to a break when macro-stepping a subroutine.

Executing a GOSUB causes the emulator to begin running. If the "L" option is set, the display will

be rebuilt with the emulated LCD image. If the "L" option is clear, the debugger will display the PC at the GOSUB address, as follows:

```
PC:3A14C:  *RUN*
```

If trace is in effect (TRC count is nonzero), the emulator will break and display the Trace Show register after the specified number of instructions.

If you macro-step into a subroutine and a break occurs within the subroutine for any reason (including "-ATTN-"), the break flag on the RSTK level is lost. If you want to restore the break at the same return address, you must set the RTN Level indicator in the #BPs register or set a break point in a Break Point register.

When macro-stepping, there are five common reasons for the emulator to be interrupted:

- For a non-GOSUB instruction, the emulator will single-step and result in an AC code of "S STEP".
- The flagged RSTK level is popped off with a RTN-type instruction, resulting in an AC code of "M STEP".
- The flagged RSTK level is popped off with a "C=RSTK" instruction, also resulting in "M STEP".
- The flagged RSTK level is pushed off the RSTK by too many "GOSUB" instructions (or "RSTK=C"), resulting in "X RSTK".
- Another break occurred in the subroutine, such as "-ATTN-", "BRK PT", "SHUTDN", etc.

Running

Press [RUN] to run the emulator. This causes the emulator to execute instructions continuously, until [ATTN] is pressed or until another reason for breaking. Setting the "K" option will alert you with a beep when the emulator stops running, except for an [ATTN] or TRACE break.

If the "L" option is set, the display will be rebuilt with the emulated LCD image. If the "L" option is clear, the debugger will display the PC at the starting address, as follows:

```
PC:3A14C:  *RUN*
```

This will remain in the display until one of the following occurs:

1. The emulator breaks and displays a register.
2. A trace display is generated (Break Point or Trace with Action=Cont).
3. The emulated routines write to the display, which you will see happen in "slow motion."

If trace is in effect (TRC count is nonzero), the emulator will break and display the Trace Show register after the specified number of instructions.

BASIC Main Loop

The BASIC main loop — entry point MAINLP, at address 002FD — is a good place to start emulation, since it requires almost no special entry conditions or register contents. Emulating from the MAINLP address will not disrupt the operating system, for, in fact, this is the entry point called to recover from system disruptions. (Some locations in memory are crucial for recovery, such as CMOSTW or the configuration table. If they have been corrupted, a Memory Lost may be

inevitable. Refer to the HP-71 IDS, Volume I, for details.)

If you are just becoming familiar with the emulator, you will find the MAINLP entry point useful for following examples. At any time during a debugging session, you can reset the debugger to MAINLP by changing the PC address to 002FD, and setting HEX mode in the PCcy register. You can single-step or run from this address.

Controlling Emulator Breaks

The PC register always contains the address of the *next* instruction to be executed. When the emulator halts, regardless of the reason, the instruction at the PC address has not been executed yet. The address in XC points to the last instruction executed.

Break Points and the PC

The emulator detects breaks before or after executing the affected instruction, depending on the type of break. The table below describes this action, according to AC code.

AC Hierarchy

The Action Completed code, shown in the XC/AC register, reflects the most significant reason for break. Since any one instruction may have several break actions associated with it, the AC code is chosen according to the following hierarchy ('A' being the highest).

AC Code	AC Hierarchy								Break before or after executing the affected instruction:
BRK PT (*1)	A	.	.	.	.	.	.	.	before
BRK PT (*2)	.	B	.	.	.	.	.	.	before
X RSTK	.	B	.	.	.	.	.	.	after
RTN BRK	.	B	.	.	.	.	.	.	after
IN PROMPT	.	B	.	.	.	.	.	.	before
UNCNFG	.	B	.	.	.	.	.	.	before
RESET	.	B	.	.	.	.	.	.	before
M STEP	.	B	.	.	.	.	.	.	after
S STEP (*3)	.	.	C	.	.	.	.	.	after
SHUTDN/PC=0	.	.	.	D	.	.	.	.	before
SHUTDN	.	.	.	.	E	.	.	.	before
BRK PT (*4)	.	.	.	.	.	F	.	.	after
TRACE	.	.	.	.	.	.	G	.	after
-ATTN-	.	.	.	.	.	.	.	H	after

Notes: (*1) Break point at first nibble of instruction.
(*2) Break point for DAT0/DAT1 memory access operation.
(*3) When single-stepping, BRK PT interrupts are ignored.
(*4) Break point at other than 1st nibble of instruction.

For example, if while running the emulator, a "RTN" instruction was encountered which had the following break actions associated with it:

1. A RTN Level indicator break;
2. A break point set on the first nibble;
3. The [ATTN] key had been hit;

then the emulator would break and report "BRK PT" for the AC code.

Note that AC codes at the same level of hierarchy are mutually exclusive, because each is caused by a different type of instruction or action.

Effects of Options

Some of the options specified in the OPTIONS register have interactions and side effects which should be taken into consideration when running the emulator. These are described in this section.

Effects of P Option

Setting the "P" option causes the PC address to be displayed whenever a break point or trace display occurs during emulation. The address will be displayed along with the Show register that you have selected. For instance, suppose a Break Point register reads:

```
BP 1~3AB19   (ST   )Cont
```

If the emulator encounters an instruction at this address, it will display first the PC, then the ST register:

```
PC:3AB19:
ST:0110 0101 0100 0000
```

Since the debugger does not have a "DELAY" time, this is most useful with an HP-IL display device.

The debugger does not automatically display the PC address after an AC code of REENTER (caused by executing **DEBUG** *, using the assembly language REENTR address, or after skimming the configuration routine). If you want the PC address displayed (or the 80-column display rebuilt) in these instances, the "P" option must be set.

Effects of K Option

Setting the "K" option causes a beep at any emulator break other than "-ATTN-", "S STEP", or "TRACE". There are no special considerations for this option.

Effects of O Option

Setting the "O" option causes an emulated "OUT=C" or "OUT=CS" instruction to be suppressed; that is, the real OUT register is not written to. With the "O" option clear, you will hear emulated beeps (greatly slowed down). But one serious effect of running the emulator with this option clear is that, when the ((•)) annunciator is on (the mainframe key buffer receives keys), multiple

interrupts will occur when a key is held down.

> **Example:** (This refers to the [ON]g key buffer toggle, explained in section 7, "Back and Forth from BASIC.") Clear the "O" option in the OPTIONS register. Set the debugger's PC to the MAINLP address, 002FD. Then press [RUN] to run the emulator at the BASIC main loop. While it is running, toggle the ((•)) annunciator on with [ON]g. Now press and hold the f-shift key for several seconds. You will notice the f annunciator flicker off and on.
>
> Now, with the f annunciator off, press and hold a character key — [X], say — for several seconds. Halt the emulator by turning off the ((•)) annunciator (with the [ON]g keystroke) and pressing [ATTN]. Now press [J][J] (with the PC register in the display) to perform a hard jump into the BASIC main loop. You will see the "X" character displayed as if you pressed the key several times.
>
> These effects occur because the emulated keyscan routines do many "OUT=CS" instructions, which, since the "O" option is clear, are written to the real OUT register. Since a key is down, this causes multiple interrupts and continuously detects the key. Each time the key is detected it is placed in the operating system's key buffer.

If this is a problem, you should either set the "O" option or not use the [ON]g toggle during emulation.

Effects of S Option

Setting the "S" option causes the emulator to break at a "SHUTDN" instruction. However, regardless of the "S" setting, a break will occur at a "SHUTDN" if the low bit of the OUT register is zero. When this happens, the debugger will beep and display the AC code "SHUTDN/PC=0".

Effects of L Option

Setting the "L" option causes the emulated LCD image to be rebuilt in the display when invoking the emulator. When first activated, the debugger stores the LCD image in RAM, where it is retrieved for emulation if "L" is set. Regardless of the "L" setting, the mainframe *display buffer* is not disturbed by the debugger, only the LCD.

When "L" is clear, the display is used by the debugger to indicate the status of the emulator. If you run the emulator (or macro-step into a subroutine), the debugger immediately displays the PC address, as follows:

```
PC:3A41C:  *RUN*
```

When "L" is set, the debugger does not display this message. The LCD image is retrieved from RAM each time the emulator is invoked (with [S], [M], or [RUN]) and copied back to RAM whenever the emulator is interrupted. During a periodic register display (such as a Break Point or Trace with Action=Cont) the LCD image is first stored, the debugger displays its registers, then the LCD image is restored before emulation continues.

The "L" option also affects the f[DISP] keystroke. Refer to the description in section 4, "Using the Keyboard."

Effects of C Option

Setting the "C" option causes the emulator to skim the configuration routines by performing a hard jump, then reentering by means of the configuration poll. While this jump is in progress, the VECTOR address is cleared (it would be disastrous otherwise, since IRAMs are unconfigured), so any keys are detected and stored by the mainframe, not the debugger. The keyboard interface is restored in less than .3 seconds, including the ((•)) annunciator status. The configuration jump action is accompanied by a short beep.

Effects of R Option

Setting the "R" option causes the debugger to issue a service request poll when it is idle and waiting for keys. The two main purposes for this are to implement the KEYBOARD IS interface and to keep the clock system current. However, in the debugger this poll is sent out between emulated instructions, when the state of memory is unknown. Poll handlers would expect that the operating system is stable, and this could cause problems. In some cases, the "R" option may have to be clear and the accuracy of the clock system will have to be sacrificed.

Effects of B Option

Displays generated by the emulator might be adjusted for BIAS, if the the "B" option is set, the register is an address register, and the address is within the adjustment range. Otherwise, the "B" option has no effect on the emulator.

Effects of H Option

Setting the "H" option causes the debugger to send displays to HP-IL (you must first have specified a DDISPLAY IS device). There is a noticeable increase in response time when "H" is set (especially if "V" is set, too) because of the additional processing. This slows down the keyboard action as well as the emulator.

Section 8, "Using HP-IL," goes into detail on the possible problems and best use of HP-IL with the debugger.

Effects of E Option

The "E" option does not affect displays generated by the emulator, such as after a single-step or break point. These are always sent to the HP-IL device, contingent upon the "H" option setting.

Effects of V Option

Setting the "V" option causes all debugger registers to be displayed with the 80-column video interface. The most noticeable effect is to greatly slow down execution of the emulator, especially if you are single-stepping or you specify frequent displays. Section 8, "Using HP-IL," describes the 80-column interface.

Effects of D Option

Setting the "D" option causes the PC register to display disassemble mnemonics, instead of nibbles, from the PC address. There are no special considerations for this option.

Instruction Set

The debugger emulates all instructions shown in Chapter 16 of the HP-71 Software IDS, Volume I. Future versions of the Saturn CPU may include new, additional opcodes. Although the HP-71 operating system will not use these, custom assembly language routines might. The debugger will not emulate properly any opcodes not shown in the instruction set of Chapter 16 of the IDS. However, since the debugger is RAM-based, advanced programmers can modify the code to handle new opcodes. Some details for modifying the debugger software are provided in section 9.

Encountering Problems

If the debugger does not respond to keys while you are running the emulator, check the following possibilities:

1. Make sure the ((•)) annunciator is off if you want the keys to be detected by the debugger rather than the operating system. Refer to section 7, "Back and Forth from BASIC," for instructions on how to use the [ON]g toggle to control the key buffers.
2. If you are using an HP-IL device, you may have caused the loop to "hang up" by trying to access it simultaneously from both systems. Refer to the subsection "Recovery from Errors," in section 8.
3. If you have been working with any of the DBGLEX files in MAIN RAM, it is likely that some operation has moved the file chain and disrupted the debugger system. Try to deactivate the debugger (using the RECOVER: 2 sequence described in the next section) and reenter using the **DEBUG** statement.
4. If none of the above methods help in regaining keyboard response, it is possible that your emulated assembly language routines have corrupted the RAM-based debugger files. If this is the case, the only recourse is to purge the DBGLEX files from memory and recopy them from your backup medium. (Remember to deactivate the debugger before purging any DBGLEX files.)

Section 7

Back and Forth from BASIC

The debugger system offers several methods for moving between operating environments. The most powerful of these allow you to work in the BASIC operating system and switch to emulation in the debugger in mid-execution.

The following methods are provided for accessing the debugger from BASIC. The keywords are described in section 3, "Setting up the Debugger." The other methods are described below.

- The **DEBUG** statement activates the debugger and waits for user control from the keyboard.
- The **DEBUG** * statement also activates the debugger, starting emulation automatically.
- The **RECOVER: 0** key sequence passes all CPU register values to the debugger so it can continue processing in emulation.
- The REENTR entry point can be called from assembly language.

The following methods are provided for accessing the BASIC environment from the debugger. All require that the debugger has been activated with the **DEBUG** statement. The methods are described in the following paragraphs.

- [Q][Q] drops you into the BASIC environment.
- [J][J] performs a hard jump to the PC address.
- The **RECOVER: 2** sequence deactivates the debugger and returns control of the HP-71 to the operating system.
- The **RECOVER: 3** sequence deactivates the debugger and clears all of memory, causing a Memory Lost. Control of the HP-71 is returned to the operating system.
- A Break Point Action or Trace Action selection of "Jump" causes the emulator to perform a hard jump to the PC address.
- The [ON]g keystroke is used to direct keys to the mainframe's key buffer or to the debugger's key buffer.

Separate Key Buffers

The debugger uses its own key detection routines and key buffer. When the debugger is activated, keystrokes are stored in its own key buffer. From there, the debugger removes and acts upon each key for direction.

Having its own key routines means the debugger takes over keyboard interrupts. This allows you to set up execution in the BASIC environment, then interrupt the operating system so the debugger can recover all CPU register information and take over processing in emulation.

Controlling the Key Buffers

Normally, the choice of where to put a keystroke — into the mainframe's key buffer, or into the debugger's — is made automatically by the system. But during a debugging session there are several actions you can take to force keys to be sent to the mainframe. Then the operating system (whether emulated or real-time) will process them. Any time this condition is in effect (the debugger sending keys to the HP-71 key buffer), the ((•)) annunciator is lit in the display.*

One time when you want to feed keys to the HP-71 operating system is when you drop into the BASIC environment from the debugger. This is the topic of the next subsection.

The other time you may want to send keys to the HP-71 operating system is when you run the emulator and want emulated routines to detect keys.

The [ON]g Toggle

When within the debugger, pressing [ON]g causes the ((•)) annunciator to toggle. When it is *on*, keys are sent to the HP-71 key buffer; when it is *off*, keys are sent to the debugger's key buffer. The annunciator (and the keyboard action) can be toggled at any time the debugger is active, except when you have dropped into the BASIC operating system. In this case, the annunciator will remain on at all times.

The [ON]g keystroke must be performed as follows:

1. Press the [ON] key first.
2. Press the g-shift key while still holding down the [ON] key.
3. Release both keys.

The ((•)) annunciator will toggle, and as soon as the g-shift key is detected, processing will continue (even though the [ON] key is still down). This processing is done in the interrupt handler, so no key code is generated by this action. [ON]g is not the same keystroke as g[ON]; in fact, the debugger does not interfere with any g[ON] key definition when you are in the BASIC environment.

> **Example:** Set the debugger's PC to the BASIC main loop address, 002FD. Before you run the emulator, turn on the ((•)) annunciator by pressing [ON]g. Now send some keys into the mainframe key buffer — press PI*2 [ENDLINE], for example. (The characters will not show up in the display yet because the operating system is suspended.) Turn off the ((•)) annunciator with [ON]g, then press [RUN] to run the emulator. The operating system will find the keys in its buffer and process them. After several seconds the characters "PI*2" will appear and the expression will be evaluated in emulation.

Dropping Into BASIC

When within the debugger, you can manually drop into the HP-71 BASIC operating system with

---

* You can select a different annunciator for this purpose if this conflicts with your application. Refer to section 9, "Additional Features and Operating Details."

four methods. Two of the methods leave the debugger active so you can resume emulating. The other two methods deactivate the debugger, returning the computer to the control of the operating system.

The HP-71 operating system, of course, does not detect the debugger break points. Only the emulator can do this, so you must reenter the debugger for break point capability.

[Q][Q] to Main Loop

Pressing [Q][Q] (the [Q] key twice within 1½ seconds) drops processing into the BASIC main loop, essentially performing an "INIT: 1". From here you can enter BASIC commands, run programs, change HP-IL devices, etc. The ((•)) annunciator remains on to signal that any keys you press will be put into the HP-71 key buffer for the mainframe to process.

You can remain in this state indefinitely, if desired, although since interrupts are being vectored into DBGLEX1A, any change to the DBGLEX files may cause a Memory Lost. For this reason you should not drop into BASIC if your DBGLEX files are in MAIN RAM.

The effect of [Q][Q] on the HP-71 operating system is identical to that of an "INIT: 1". This includes turning off User mode and reconfiguring all memory devices.

[J][J] for Hard Jump

Pressing [J][J] (the [J] key twice within 1½ seconds) performs a hard jump to the PC address. All restorable registers in the CPU are restored from their emulated counterparts and execution picks up at the PC address. (A list of restorable registers is shown under "Hard Jump to PC" later in this section.)

[J][J] can only be executed when the PC or PCcy register is in the display.

As with [Q][Q], the ((•)) annunciator is lit to indicate that keys are being processed by the mainframe, although the debugger is active. You can remain in this state indefinitely, if desired, with the risks mentioned above.

RECOVER Methods

The other two methods of returning to BASIC are described below, under "RECOVER: 2" and "RECOVER: 3".

Reentering the Debugger

If you are in the BASIC environment — including not having yet activated the debugger — you can always execute the BASIC statement **DEBUG**. This will re-establish all emulated CPU registers to the contents held at the time you last left the debugger, except for status bits S15-S12 and any RAM contents that might have been changed in BASIC.

If you have dropped into BASIC with [Q][Q] or [J][J], the "RECOVER: 0" sequence will be most useful for reentering the debugger. As described below, this sets up the debugger for emulation at the point of the [ON][/] interruption.

There is a third way to reenter the debugger: through an assembly language jump to the debugger reenter address. This is described in the next subsection.

The RECOVER Sequence

The familiar INIT message in the HP-71 mainframe is replaced by the RECOVER prompt. When the debugger is active, pressing [ON][/] (both keys simultaneously) causes the prompt "RECOVER: 1". You can select four levels of recovery by pressing one of the digit keys [0] through [3], followed by [ENDLINE].

You must perform the [ON][/] keystroke with the following sequence:

1. Press the [ON] key first.
2. Press the [/] key while still holding down the [ON] key.
3. Release both keys.

If you press [/] first, you may cause an "INIT": 1" instead of a "RECOVER: 1" since the operating system may detect both keys in a synchronous keyscan.

RECOVER: 0

When operating in the debugger, "RECOVER: 0" is not allowed. The computer will beep and reprompt for the RECOVER level.

When in BASIC with the debugger active — you get there by pressing [Q][Q] or [J][J] from within the debugger — the "RECOVER: 0" sequence interrupts the operating system and copies all recoverable CPU registers into their emulated counterparts. At this point the debugger displays the PC register and waits for manual direction. (A list of restorable registers is shown under "Assembly Language Reenter" later in this section.)

The "RECOVER: 0" sequence is a powerful method of directing the debugger. With precise timing you can interrupt the real-time execution of a target routine and set up all emulator registers for debugging.

> **Example:** You have written a new keyword, "SCALE", which calls the math routine IDIV in the mainframe (address 0EC7B) to perform a division. You suspect that the registers are not set up properly for entry into IDIV, and you want to examine them when that routine is called. Perform the following steps:
>
> 1. Execute **DEBUG** to enter the debugger. Select the Break Point stack and enter the address of IDIV, 0EC7B, in the first available break point register. Now press [Q][Q] to drop into the HP-71 main loop.
> 2. Type **SCALE** to execute the keyword, but don't press [ENDLINE] yet.
> 3. Now press [ENDLINE], and quickly press [ON][/] to interrupt the operating system and get the RECOVER prompt. Press [0] to select "RECOVER: 0", then [ENDLINE].
> 4. The debugger is now ready to use, with all registers, including the PC, at the point of the [ON][/] interrupt. Press [RUN] to run the emulator, and soon the break point at 0EC7B will be encountered. (If the interval between [ENDLINE] and [ON][/] was too long, you might have already passed the 0EC7B address. You should examine the PC before you press [RUN] to make sure you haven't passed the break point. You will soon get the feel of how long an interval is needed to get close to the target address without passing it.)

RECOVER: 1

When within the debugger, the "RECOVER: 1" sequence returns you to a stable point, displaying the PC register.

When you are in BASIC with the debugger active — after [Q][Q] or [J][J] — the "RECOVER: 1" sequence performs an "INIT: 1", returning you to the stable point of the BASIC main loop.

In either case, a "RECOVER: 1" is not guaranteed to be a safe action. If memory is being altered or moved at the time of the [ON][/] interrupt, the system may not be able to recover from the disruption and cause a Memory Lost. "RECOVER: 1" (and, for that matter, "INIT: 1") should not be performed lightly.

RECOVER: 2

This deactivates the debugger, returning complete control to the BASIC operating system, including re-establishing keyboard control and the INIT prompt. The effect on the BASIC system is identical to performing an "INIT: 1". To reenter the debugger you must execute the **DEBUG** statement.

When you are done with a debugging session, you should routinely perform a "RECOVER: 2" to deactivate the debugger. This will prevent problems in the case (unlikely, unless the files are in MAIN) that the DBGLEX files are moved or changed.

RECOVER: 3

This performs an "INIT: 3" — a Memory Lost, deactivating the debugger in the process.

Hard Jump and Assembly Language Reenter

Two powerful features of the debugger allow you to access assembly language from the debugger and vice versa. You can use the debugger to examine registers in mid-execution of a routine. Or you can make adjustments to register contents and exit through the routine in real-time.

Example: You want to determine the status bit settings at a certain point in the execution of your new keyword, "SCALE". You can do this in the debugger without keyboard intervention by "programming" with the following steps. Suppose the address of your routine is 3A14C, the location where you want to display the ST register.

1. Activate the debugger with **DEBUG** and select the Break Point stack.
2. Edit the first available Break Point register (assume here BP 1) to show:

```
BP 1~3A14C  (ST   )Jump
```

3. Drop into the BASIC environment with [Q][Q], and enter

```
DEBUG * @SCALE   [ENDLINE]
```

The debugger will begin emulation automatically. When the emulator encounters the break point at 3A14C, it will display the ST register and exit through your routine by performing a hard jump to 3A14C.

Example: You are in the debugger emulating a BASIC program, and you want to skip through a FOR/NEXT loop to the next occurrence of the "SCALE" keyword. Instead of having the emulator run through the loop, at its slow speed, you can perform a hard jump into it and pop back into the debugger with RECOVER: 0 for the "SCALE" execution. (Timing is important here, in order to catch the processor before execution of the "SCALE" keyword.)

Hard Jump to PC

Pressing [J][J] when the PC is in the display, or executing the "Jump" Action for Break Point or Trace will cause the debugger to perform a hard jump to the PC. This causes real-time execution of the code at the PC address. You can reenter the debugger with the "RECOVER: 0" sequence, with the **DEBUG** statement from BASIC, or with an assembly language reenter (see below).

A hard jump to the PC causes the ((•)) annunciator to be lit, indicating that keys are sent to the operating system's key buffer. Whereas [Q][Q] performs an "INIT: 1" and turns off User mode, the [J][J] keystroke does not. What effect it has on the operating system depends entirely upon the routine into which you jump.

Before jumping to the PC address the following restorable registers are copied from the emulator to their CPU counterparts (these also apply to the "Jump" action for a Break Point register or TRC):

Registers restored:

A B C D
R0 R1 R2 R3 R4
First 7 levels of RSTK (L0 - L6)
ST
HS bits: SB and XM only
OUT
Interrupt
D0 D1
Carry, Pointer, Mode
PC

Registers not restored:

8th level of RSTK (L7) (set to zeroes)
HS bits: SR and MP (retain their current real CPU values)

Assembly Language Reenter

Your assembly language routine can jump directly back to the debugger without using the **DEBUG** statement or the manual "RECOVER: 0" sequence. The entry point "REENTR" in the debugger is for this purpose.

To use the "REENTR" feature you must have a strategically placed "GOSBVL REENTR" in your RAM-based assembly language application. You need to first compute the actual address of the REENTR label according to the address of DBGLEX1A in memory. You can specify the address in your assembly source file, or fill it in afterwards with a POKE or by memory editing in the debugger.

Appendix C contains a list of offsets to important entry points in the DBGLEX files. That appendix also includes directions for computing absolute addresses of entry points.

Example: You want to examine the execution of your new keyword, "SCALE", at a certain point in the routine, starting at "LABEL3". Your assembly language code should look like this (the four instructions marked "*" are for example only):

```
          A=C      A        *    Opcodes
          CD1EX             *     as desired...
          GOSBVL   #00000        Fill in address of REENTR !
LABEL3    B=B+C    A        *    More opcodes
          D0=A              *     as desired...
            .
            .
            .
```

Follow this procedure:

1. Assemble and load the LEX file in memory.
2. Compute the address of "REENTR" as described in appendix C and remember the address.
3. Now enter the debugger by typing **DEBUG** — this sets up some addresses and pointers required for later reentry. Set a window register to the address of the "GOSBVL" in your file. Using f[EDIT] to edit memory, fill in the address at that opcode (remember to reverse the five nibbles within the "GOSBVL" instruction). The instruction is now a

"GOSBVL REENTR".

4. Press [Q][Q] to drop into BASIC. Now type the keyword **SCALE** and press [ENDLINE]. The debugger will take over at the "GOSBVL REENTR" with the PC displayed (it is the address of "LABEL3"), waiting for direction.

When reentering the debugger, the emulator's CPU registers are recovered from the real CPU registers. The recoverable registers, which also apply to the "RECOVER: 0" sequence, are:

Registers recovered:

A B C D
R0 R1 R2 R3 R4
First 7 levels of RSTK (L0 - L6) (see note below)
ST
HS bits: all (MP, SR, SB, XM)
D0 D1
Carry, Pointer, Mode
PC (see note below)

Registers not recovered:

8th level of RSTK (L7) (set to zeroes)
OUT (retains last debugger value)
Interrupt (retains last debugger value)

The following registers are untouched; they retain their values from when the debugger was last exited:

User's stack
Window stack
Break Point stack
Options stack

In addition, the XC register is reset to 00000 and the two RSTK break indicators (the macro-step indicator and the RTN Level indicator) are cleared.

The AC code after a "RECOVER: 0" is "-ATTN-". The AC code for a reentry through the REENTR address is "REENTER".

The RSTK levels recovered are actually levels L0 through L6 *at the moment before the [ON][/] interrupt* (in the case of "RECOVER: 0") or *at the moment before "GOSBVL REENTR"* (in the case of assembly language reenter). When the debugger is reentered with either of these two methods, the PC is set to the address of the return (from interrupt, or from REENTR) and level L7 is lost. This is why, in the example above, a "GOSBVL REENTR" is used to set the emulator's PC to LABEL3.

You can have more control over the debugger when using the REENTR entry point by setting options in certain RAM locations. Refer to section 9, "Additional Features and Operating Details."

Section 8

Using HP-IL

Debugging sessions are more productive when you can view several registers at once. The debugger allows you to specify an HP-IL display device for its output. The device can be a video interface, a printer, or another character-oriented device, such as an HP-IL/RS232 interface.

You should not run the debugger with the HP-IL ROM copied into RAM if you plan to use a debugger display device. Doing so will result in a Memory Lost when the HPILROM file moves in memory.

The Debugger Display Device

In order to use an HP-IL display device, you need to execute the DDISPLAY IS statement and set the "H" option in the OPTIONS register.

The DDISPLAY IS Statement

The file DBGMAINA provides the BASIC statement DDISPLAY IS, which is used to specify the debugger display device. The device is encoded and stored in the debugger's RAM, so once you specify the DDISPLAY IS device you do not have to respecify it, even if you deactivate the debugger. However, if you change the loop by removing or reordering the devices, you should execute DDISPLAY IS again.

The DDISPLAY IS statement will execute only if the HP-IL interface is plugged in. If the module is missing, the statement will cause the error "ERR:XWORD Not Found".

You should execute DDISPLAY IS in BASIC before entering the debugger. If not specified, the device defaults to "DDISPLAY IS *", and all debugger output is sent only to the LCD. Statement syntax is identical to the DISPLAY IS keyword in the HP-IL module, except that certain device specifiers are not allowed.

The following device specifiers are allowed for DDISPLAY IS (refer to page 75 of the *HP-IL Interface Owner's Manual*):

- Accessory type. Example: DDISPLAY IS %48:2
- Device word. Example: DDISPLAY IS RS232
- HPIL address. Example: DDISPLAY IS 4
- Assign code. Example: DDISPLAY IS ":TV" (after executing an ASSIGN IO statement which assigns ":TV" to a device).

Certain device specifiers cause a system buffer to be created and searched at display time. These are not allowed by DDISPLAY IS because during emulation the unknown state of memory

prohibits searching for system buffers. The following device specifiers are not allowed for DDISPLAY IS:

- Device type. Example: DDISPLAY IS :HP82905B
  Causes: DBG ERR:Illegal Device Spec
- Volume label. Example: DDISPLAY IS .STORE
  Causes: DBG ERR:Illegal Device Spec

As with the HP-IL command DISPLAY IS, you can cancel the debugger's HP-IL display device with the following equivalent forms of DDISPLAY IS:

- DDISPLAY IS *,
- DDISPLAY IS "*", or
- DDISPLAY IS ""

These forms of the statement also have another important effect: they disable the debugger HP-IL by clearing the "H" option and the display handler address. The reasons for this are twofold. You should execute DDISPLAY IS * :

1. Whenever you remove the HP-IL module from the HP-71 with the debugger in memory. The proper way to do this is to deactivate the debugger, execute DDISPLAY IS *, then remove the HP-IL module. You can then reactivate the debugger, if desired. Failing to execute DDISPLAY IS * before removing the HP-IL module may disrupt the system and cause a Memory Lost when you reenter the debugger.
2. When attempting to recover from an HP-IL error generated from within the debugger. More details on this are given below.

Sharing a Display Device. You can specify the debugger's display device as the same one used by the BASIC operating system. However, there are some precautions you should take.

Occasionally the HP-IL interface may become confused because the debugger and the emulated routines try to access the loop simultaneously. The result may be to swap display devices or to "hang" the loop (from which only a RECOVER:2 will recover). This can happen if your debugging session causes frequent displays (such as single-stepping or tracing) and you are emulating in the HP-IL module. The best approach in this case is to clear the "H" option to avoid the problem.

Running the emulator through such routines should not cause problems as long as debugger displays are not sent out periodically. Potential problems occur only when the emulated routines send control frames on the loop, followed by characters from the debugger display routines.

Separate Display Devices. You can share one loop between the operating system and the debugger, while specifying different display devices for each. This makes it much easier to distinguish output from the two systems, but you may still experience problems if they both try to access the loop simultaneously during emulation.

Dedicated Loop. You will get best results with the debugger display device if you use the Dual HP-IL Adapter (part number HP 82402A) and dedicate one loop specifically for debugger output. This will completely separate the debugger's actions from the emulated routines and minimize the possibility of loop disruption.

Alternatively, if you are using only one loop, you can emulate the operating system with OFF IO and use the loop exclusively for debugger output.

Specifying a Printer. If you designate a printer for the debugger display device, you may want to modify the debugger's output routines to avoid overstrike. Otherwise, when editing registers, the printer will perform many carriage returns and print over the current line. The debugger has three

fields similar to the ENDLINE string in BASIC for controlling the output device — except that the debugger's fields are sent out *prior* to displaying a line. Modify the "EdtLin" field, as directed in appendix C, to prevent overstrike. The "NewLin" and "VdoLin" strings can be changed, too, to format the output as you like.

Recovery From Errors

If you are sharing one loop between the debugger and the operating system, you will probably experience a "lockup" eventually, when both systems try to access the HP-IL mailbox simultaneously. This situation is avoidable by following the precautions described in the previous paragraphs.

When lockup occurs, the debugger will not respond to keys except for the RECOVER sequence. RECOVER: 1 will not restore the PC register since HP-IL again hangs during display. The first step to take is to use the [ATTN][ATTN] interrupt feature of the HP-IL interface. The HP-IL routines which detect this keystroke use the mainframe key buffer.

1. Toggle on the ((•)) annunciator with [ON]g .
2. Press [ATTN] twice.
3. Toggle off the ((•)) annunciator with [ON]g .

The debugger's keys should again be active. You may need to execute the "Restore Debugger IO" command in the extended command menu to regain the HP-IL display. Refer to section 9, "Additional Features and Operating Details."

If the above steps do not recover control of the debugger, you need to deactivate the debugger to restore the HP-IL interface. Use the following procedure:

1. Perform the RECOVER: 2 sequence to deactivate the debugger.
2. Execute DDISPLAY IS * to clear the debugger HP-IL interface.
3. Activate the debugger with the **DEBUG** statement. The debugger should now be active and respond to keys. Only the LCD will be used for displays since the DDISPLAY IS * statement cleared the "H" option. You can set it again at this point.
4. You can drop into the BASIC environment now to execute the DDISPLAY IS statement, specifying the desired device.

If the debugger HP-IL is not restored with this procedure, you should repeat the process, executing RESTORE IO before step 2 (remember to specify the correct loop number if you are using the Dual HP-IL Adapter). If you still have difficulties, it may be necessary to execute RESET HPIL, also.

Rarely, the HP-IL module may issue an error while under the control of the debugger (this might happen if you break the loop in the middle of the 80-column display, for example). Since errors return to the BASIC environment, this will put the debugger system in a strange state — the operating system has taken over without control of the keyboard. If an HP-IL error occurs during a debugger display and the keyboard responds sluggishly, you should perform a RECOVER: 1 to restore the debugger. You should also check the integrity of the loop; you may need to use the "Restore Debugger IO" command in the extended command menu (refer to section 9).

The 80-Column Video Interface

If you have set up the debugger to use an HP-IL display device, you can have all CPU registers displayed in an 80-column template. You can use any 80-column device for this purpose, such as a Mountain Computer™ HP-IL 80-Column Video Interface, an HP-IL/RS232 interface with a terminal device, or a printer.

To enable the 80-column display you must specify a display device with DDISPLAY IS, set the "H" option (to send displays to HP-IL), and set the "V" option (which initiates the 80-column interface).

An illustration of the 80-column register template is shown in Figure 3. All CPU registers (except I — interrupt Enable/Disable) are displayed, along with the first seven Break Point registers and the first four Window registers.

The first 10 lines of the display device are used for the 80-column template. The remaining lines are used for normal register displays, such as moving around the stacks. When you are working in the debugger the 80-column template is updated whenever:

1. You press the [ENDLINE] key; or
2. The emulator is interrupted to display registers (such as after a single-step, halting for [ATTN], or for a trace display).

For example, as you single-step through a routine, after each [S] keystroke the template is updated to show the effects of the single-step. If you select the PC as the current register and set the "D" option, after each single-step the PC will be displayed, with its mnemonic, underneath the 80-column template. In this manner you can display a history of the last 14 or so instructions.

The 80-column template is separate from normal register displays. That is, moving around in the register stack does not cause the template to be updated. The "E" option has the same effect whether or not the 80-column display is in effect: if "E" is set, normal register displays are not sent to HP-IL except when you press [ENDLINE] or when generated by an emulator break.

The 80-column display significantly increases the response time for keyboard input as well as slowing down emulation, since much time is spent in rebuilding the display. You may want to clear the "H" or "V" options occasionally to speed up a routine, or during register manipulations.

The default control string for the video interface assumes your device has 24-line by 80-column capability. If your display has less than 24 lines, you can modify this action by changing the "VdoLin" string to fit your device. Refer to section 9 for details. That section also describes the extended command menu, which contains some commands for controlling the HP-IL device.

Mountain Computer is a trademark of Mountain Computer, Incorporated.

```
-------------------------------------------------------------------------------
PC:43C9E  C:0 P:4 M:H    A:000000000000000A  R0:0270000000000001  0:43681   5 BP
   ?A=0 A/GOYES 43CBB    B:0000000000001C44  R1:0000002FAB737FF6  1:01C72  13665
XC:43CB8: -ATTN-         C:02FA0300020FF000  R2:0000000000000005  2:01C48  1BBAD
                         D:00050000000F2401  R3:0275000000000001  3:00000  2F78D
2F599.0008300083000830                ^      R4:0C0000000F000063  4:00000  3C801
2F599<0000000000000000   D0:37FF4: 022370D8                       5:00000  43C9E
1BBAD.76C559D240D7F855   D1:2FA5D: 100F0000   1000 1101 0111 1011  6:00000
3C801#0000000000000000   OUT:000 IN:0000(0)   MP:0 SR:0 SB:0 XM:0  7:00000
-------------------------------------------------------------------------------
```

Explanations:

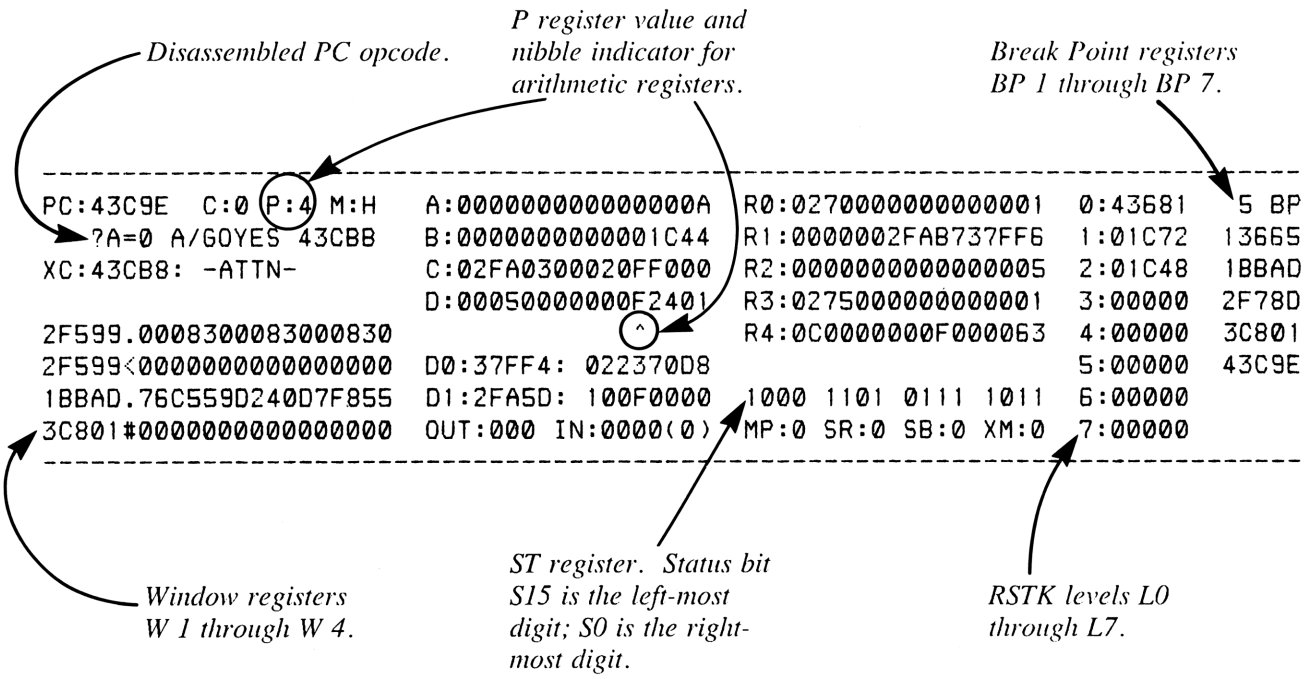


Figure 3. Debugger 80-Column Display

Using a Keyboard Device

A KEYBOARD IS device can be used with the debugger in some situations. This requires an HP-IL interface, a remote keyboard, and the LEX file "KEYBOARD", available from the HP-71 User's Library or found in the FORTH/Assembler ROM and HP-IL LINK (part number HP82477A). The "R" option must be set in order for the KEYBOARD LEX file to pass keys to the debugger.

Restrictions

There are several conditions which restrict the use of KEYBOARD IS with the debugger:

1. The debugger does not have a dedicated KEYBOARD IS device, but shares one with the BASIC operating system.
2. Setting the "R" option causes a service request poll to be sent out when the debugger expects keys. In the operating system, polls are sent out only when activity allows it. In the debugger, sending out a poll between emulated instructions may find memory suspended in a disrupted state. You should carefully consider the state of emulation before proceeding with the "R" option set.
3. The KEYBOARD LEX file uses a system buffer to store ESCAPE assignments. For each keystroke from the remote keyboard, the LEX file needs to access the system buffer, whether you have made any ESCAPE assignments or not. In between emulated instructions, memory might be suspended in a disrupted state where system buffers should not be accessed. If you are unsure of the state of memory, you should avoid using the remote keyboard.
4. The increased activity on the loop makes for a greater possibility of hanging HP-IL. If some action causes the loop to abort, you may need to drop into the BASIC operating system to execute RESTORE IO or RESET HPIL.
5. Some important control actions cannot be performed by the KEYBOARD IS device. For instance, you cannot interrupt the emulator with the mapped [ATTN] key and you cannot generate the RECOVER sequence. You need to perform these with the HP-71 keyboard.

In most debugging situations, where you have a self-contained assembly language routine, the KEYBOARD IS device will work properly. However, if your emulation involves memory movement or you are simulating a large portion of the HP-71 operating system, you may easily run into difficulties. In these cases, you should use only the HP-71 keyboard to direct the debugger.

Keyboard Without Debugger

You can always use the KEYBOARD IS device to direct the operating system, without the debugger interpreting the keys. Having dropped into the BASIC operating system with [Q][Q] or [J][J], keys are always interpreted by the mainframe only.

You don't have to deactivate the KEYBOARD IS device when you pop back into the debugger. You can clear the "R" option so that the device will not be detected by the debugger. Or, if the "R" option is set, simply don't press any keys on the device.

Key Mapping

Keys on the KEYBOARD IS device may be pressed in either upper case or lower case. The only exceptions are the [Q][Q] and [J][J] keystrokes, which need to be upper case.

All character and digit keys are mapped one-to-one with the equivalent debugger keys. Several keystrokes on the HP-71 are not available on a KEYBOARD IS device. The following key mappings have been made for debugger keystrokes. Each uses a control (CTRL) sequence.

Debugger action	KEYBOARD IS equivalent
f[AUTO]	CTRL A
f[DELETE]	CTRL D
f[USER]	CTRL F
g[1USER]	CTRL G
f[KEY]	CTRL K
f[LIST]	CTRL L
f[OFF]	CTRL O
f[PURGE]	CTRL P
[RUN]	CTRL R
g[CMDS]	CTRL X
f[EDIT]	CTRL Z

If you have copied file DBGLEX2A into RAM, you can change these mappings or add new ones. Refer to appendix C for details concerning the CTRLTB location.

Section 9

Additional Features and Operating Details

This section describes some of the features you will find useful as you become more familiar with the debugger system.

The Extended Command Menu

When the file DBGLEX3A is in memory, you have access to a menu of extended commands to control special debugger actions. Pressing g[CMDS] in view mode gains access to nine commands which can be scrolled up and down with the arrow keys. When the desired command is in the display, pressing [ENDLINE] will execute it. The extended command menu can be aborted by pressing the [ATTN] key.

The nine extended commands are:

```
Clear Display
Restore Debugger IO
Reset HPIL
Wndw + DBGADDR$(1)
Wndw + DBGADDR$(2)
Wndw + DBGADDR$(3)
Initialize Debugger
Set Checksum
Verify Checksum
```

When in the extended command menu only four keys are active:

1. [ATTN] key: aborts the command menu and redisplays the current register.
2. [↓] key: rolls the menu to the next lower entry (with wrap-around to the top).
3. [↑] key: rolls the menu to the next higher entry (with wrap-around to the bottom).
4. [ENDLINE] key. Pressing [ENDLINE] executes the indicated command and — unless indicated below — returns the display to the current register.

Clear Display

This sends escape sequences to the HP-IL display device to clear the screen. (The escape sequences are: Escape H, Escape R, Escape J.)

Restore Debugger IO

This attempts to restore the loop with the specified DDISPLAY IS device as well as the KEYBOARD IS device. This is not as robust as the HP-IL command "RESTORE IO", and therefore will work only in simple cases, such as after aborting HP-IL by hitting [ATTN] twice. If

"Restore Debugger IO" does not re-establish the debugger's HP-IL, you may need to execute the "Reset HPIL" command, described next.

Reset HPIL

This attempts to execute a "RESET HPIL" command for the loop specified in the DDISPLAY IS device. This is not as robust as the HP-IL command "RESET HPIL". If an error occurs, the debugger will beep and show "Reset HPIL Failed" for 1½ seconds. In this case, the only recourse may be to drop into the operating system and execute "RESET HPIL" in BASIC.

If the "Reset HPIL" command is successful, it will also perform the "Restore Debugger IO" command automatically.

Wndw + DBGADDR$(1)

If the current register (before enabling the extended command menu) is in the Window stack, executing this command will add the value returned by the DBGADDR$(1) function to the window address; the resulting address is replaced in the window register. If the current register is not a window register, the debugger will only beep and redisplay the unaffected register.

This command is useful when modifying parts of the debugger RAM map, control locations, or code. Using the addresses published in appendix C of this manual, you can set a window register to any location in the DBGLEX files. Then with RAM edit (f[EDIT]), you can modify the contents of memory at that location.

Wndw + DBGADDR$(2)

This command is identical to the previous one, except that it adds the value of DBGADDR$(2) to the window register address.

Wndw + DBGADDR$(3)

This command is identical to the previous one, except that it adds the value of DBGADDR$(3) to the window register address.

Initialize Debugger

This command resets all debugger CPU registers to zeroes, clears the User's stack, the Window stack, and the Break Point stack. It sets one break point at address 00209, the location where the VECTOR storage is cleared. This break point will offer protection from inadvertently emulating through the cold start code.

The "Initialize Debugger" command does not clear the OPTIONS stack, nor does it change the HP-IL device control locations or the display strings (EdtLin, NewLin, and VdoLin — used to control the display during output).

Executing the "Initialize Debugger" command sounds a short beep. The current register becomes the PC (set to address 00000) and the AC code is "INITLZ".

Set Checksum

This command causes the debugger to compute and store a checksum byte for each of the three DBGLEX files. The display will show each byte as it is computed; for example,

```
Set Checksum  DA B5 6B
```

This indicates that the checksums are DA, B5 and 6B for each of the files DBGLEX1A, DBGLEX2A, and DBGLEX3A, respectively. The checksum for file DBGLEX1A does not include the RAM storage area, so it is not affected by emulated CPU register contents.

Since the DBGLEX files are in RAM, they are vulnerable to bugs which clear or overwrite memory. Any time you suspect the files have been corrupted, you should execute the "Verify Checksum" command (described below) which will report checksum errors. Checksums are computed and verified only by explicitly executing these two commands.

The source copies of the DBGLEX files have the checksum bytes filled in. If you make modifications to the debugger software, you should execute the "Set Checksum" command to afford protection against corruption of the files.

Verify Checksum

This command verifies the checksum bytes for the three DBGLEX files by recomputing the checksums and comparing them against the stored values. Results are displayed for 1½ seconds. For example,

```
Verify Checksum  G G B
```

This indicates that file DBGLEX1A is "Good", DBGLEX2A is "Good", and DBGLEX3A is "Bad". If a checksum indicates "Bad", it is likely that the RAM-based software has been corrupted. In this case, it is advisable to purge the affected file and recopy from your backup on mass medium. (Remember to deactivate the debugger before purging any of the DBGLEX files.)

The Disassembler

All opcode mnemonics generated by the disassembler are the standard ones as shown in the HP-71 Software IDS, Volume I, Chapter 16. The few exceptions concern non-standard instructions involving hardware status bits, and NOPs (no-ops).

Jump instructions (GONC, GOTO, GOSBVL, GOYES, etc.) display an absolute address. The jump address will be adjusted for the BIAS if it falls within the BIAS range and the "B" option is set. GOYES and RTNYES instructions are displayed on one line with the accompanying test mnemonic. For example,

```
PC:04C5D:  ?D#C WP/GOYES  04C3D
```

For displays which exceed 22 characters, only the last 22 characters are shown in the LCD.

Hardware Status Bits

The following table lists the hardware status bit instructions. The ones which affect more than one HS bit are non-standard; these mnemonics show an "H" and the second character of the HS bit names.

Opcode	Mnemonic	Opcode	Mnemonic	HS bits cleared/tested
820	NOP820	830nn	?H0=0/GOYES	none
821	XM=0	831nn	?XM=0/GOYES	XM
822	SB=0	832nn	?SB=0/GOYES	SB
823	HBM=0	833nn	?HBM=0/GOYES	SB,XM
824	SR=0	834nn	?SR=0/GOYES	SR
825	HRM=0	835nn	?HRM=0/GOYES	SR,XM
826	HRB=0	836nn	?HRB=0/GOYES	SR,SB
827	HRBM=0	837nn	?HRBM=0/GOYES	SR,SB,XM
828	MP=0	838nn	?MP=0/GOYES	MP
829	HPM=0	839nn	?HPM=0/GOYES	MP,XM
82A	HPB=0	83Ann	?HPB=0/GOYES	MP,SB
82B	HPBM=0	83Bnn	?HPBM=0/GOYES	MP,SB,XM
82C	HPR=0	83Cnn	?HPR=0/GOYES	MP,SR
82D	HPRM=0	83Dnn	?HPRM=0/GOYES	MP,SR,XM
82E	HPRB=0	83Enn	?HPRB=0/GOYES	MP,SR,SB
82F	CLRHST	83Fnn	?HPRBM=0/GOYES	MP,SR,SB,XM

NOPs

A "NOP," or "no-op," is an opcode which has no defined operational effect on any CPU register (other than incrementing the PC). The debugger does not ignore a NOP, but executes it as it does any other opcode in its class.

The disassembler treats the following opcodes as NOPs and displays the indicated mnemonics:

Opcode	Mnemonic	Meaning
420	NOP420	GOC to next instruction
520	NOP520	GONC to next instruction
6300	NOP6300	GOTO to next instruction
818	NOP818	Unspecified SRB (shift-right-bit) instruction
819	NOP819	Unspecified SRB instruction
81A	NOP81A	Unspecified SRB instruction
81B	NOP81B	Unspecified SRB instruction
820	NOP820	Clear no HS bits.
8C4000	NOP8C4000	GOLONG to next instruction

Debugging Techniques

There are some special considerations when debugging in the HP-71 mainframe. The debugger system uses some of the features of the BASIC operating system to pass control back and forth between the environments. Assembly language routines must insure that they do not interfere with these actions.

Avoiding Cold Start

You can take precautions so that you don't emulate through the cold start code (address 00000) since these routines clear all of memory. Many assembly language bugs show up as a Memory Lost, so you will probably find the emulator jumping here occasionally.

Since the debugger uses the interrupt vector, the VECTOR location in main RAM must not be cleared. Doing so would send the debugger into oblivion, emulating without any way to detect keys. If this does happen, "INIT: 1" may bring back the computer, but "INIT: 3" might be the only way to recover. In any case, if the RAM clearing has already taken place, it may be too late to avoid a Memory Lost.

You may want to set a break point at address 00000 to prevent emulation of the cold start routine. Breaking at this address will prevent any RAM from being cleared and you can almost always reset the debugger without causing a Memory Lost.

You can provide protection from clearing VECTOR if you set a break point at address 00209, the location where VECTOR is cleared. Executing the "Initialize Debugger" command in the extended command menu will set an automatic break point at this address, after clearing all debugger registers. Thus, it is good practice to initialize the debugger at the start of any debugging session. If the debugger ever runs through the cold start section, it will halt at break point 00209 with D1=2F43C. You should not continue emulation. Usually, if you set the PC to the address of the BASIC main loop ("MAINLP", address 002FD) and set HEX mode (in the PCcy register), you can recover with no further register manipulation.

System Timers

Each of the three display chips contains a six-nibble countdown timer. When working with these system timers, special handling may be required during emulation. The CLKSPD routine (address 0E9F1) is a trap for the debugger. This routine computes the CPU speed in reference to a timer, waiting for a specific timer value to exit a counting loop. Since the debugger runs so slowly in relationship to the HP-71 operating system, it can run indefinitely in this counting loop as the timer speeds along. There are actually three locations in the CLKSPD routine where the timer is read and compared, so if you want to run the debugger through this routine you must adjust the A and C registers for each comparison. Use the HP-71 IDS, Volume III, as a guide.

Other routines which read timers (such as WRTTMR, address 15392) will not hang up the debugger indefinitely, since they do not enter a loop. However, the timer value read by the debugger will reflect the fact that it is running relatively slowly; if you are concerned about critical time intervals, you should halt the debugger at locations like this and edit a specific value into the A or C register.

Configuration

Because the debugger resides in soft-configured RAM, it cannot allow emulation of the configuration routines. The following BASIC statements or actions cause the computer to reconfigure memory:

1. FREE PORT and CLAIM PORT.
2. OFF, BYE, or the f[OFF] keystroke, then wakeup.
3. Inserting or removing plug-in modules.
4. Copying a LEX file into memory or purging a LEX file.
5. Reconfiguration forced by an assembly language application.
6. Cold start.

If you try to emulate any of the above, the debugger will encounter the "RESET" instruction at address 10233 in the configuration routine. If you have set option "C", the debugger will "skim through" the configuration code and recover with the configuration poll. If option "C" is not set, the emulator will break, beep, and display the AC code "RESET".

Unless a poll handler performs unexpected actions, there should be no problems arising from using the "C" option since the configuration poll is issued at a known location and state in the operating system. This assumes, of course, that you have not affected configuration addresses by inserting or removing modules or executing FREE PORT or CLAIM PORT. The debugger may not be able to restore properly if the configuration of the computer undergoes a change. Specifically, the addresses of the DBGLEX files may change, and if this happens a Memory Lost will be inevitable. To guard against this, you should deactivate the debugger with "RECOVER: 2" before inserting or removing plug-ins and before executing FREE PORT or CLAIM PORT.

You cannot emulate an "UNCNFG" instruction which would affect a DBGLEX file, or a "RESET" instruction at an address other than 10233. In these cases, you can either increment the PC past the instruction or perform a hard jump with [J][J] into the code and recover quickly with the "RECOVER: 0" sequence.

Emulating Poll Handlers

When you set the "C" option, the debugger intercepts the configuration poll by replacing the poll return address with its own address. Thus, if you are emulating a configuration poll handler which takes similar actions, you should not set the "C" option. To prevent reentry into the debugger when skimming the configuration routine, a pCONFG poll handler should never call the REENTR entry point.

For similar reasons, a service request poll handler (pSREQ poll) should never jump into the debugger with REENTR. The debugger "R" option causes a service request poll to be issued, so if you need to emulate such a poll handler you should not set the "R" option.

Conflicts with VECTOR

If your assembly language application uses the VECTOR interrupt RAM location, you can only emulate those parts of the application which do not depend on the VECTOR. If you alter your code to not use the VECTOR location, you can use the debugger to simulate an interrupt for testing your interrupt handler. Only when you are done using the debugger would you reinstall the VECTOR interception in your code.

Modifying the Debugger Software

If your DBGLEX files are in RAM, you can modify portions of the code or the RAM map to customize the debugger's operation. You should always keep a backup copy of the debugger source files on a write-protected medium, in case the files in memory are corrupted.

Annunciator Control

The two annunciators used by the debugger system can be changed if they conflict with your application. The defaults are:

1. The ((•)) annunciator, to indicate that keystrokes are being sent to the BASIC operating system key buffer.
2. The USER annunciator, to indicate when the User's stack is being displayed.

Each annunciator is controlled by specifying its address and bit mask (a nibble value with the required bit to turn it on). In addition, the keyboard annunciator, ((•)), is also controlled within the operating system by a system flag. (Refer to page 201 of the *HP-71 Owner's Manual* for a list of these flags.) By writing a five-nibble address and a one-nibble bit mask to the appropriate RAM locations with DBGLEX1A, you can use any annunciator or bit in the display (or combinations within one nibble).

For the keyboard annunciator, ((•)), you need to change the DBGLEX1A locations ANCTRL and ANFLAG. For the User's stack annunciator, you need to change USCTRL. Each of these locations stores an address and a mask. You can disable all annunciator usage by writing 000000 to each of these fields. The following table shows the values to put into the RAM locations for each of the annunciator selections. Some annunciators are not available to the debugger, since the operating system changes them frequently (these are indicated with "n/a" in the table). Directions for locating the RAM locations points are in appendix C, "Addresses of Entry Points."

Annunciator	User stack annunciator (USCTRL)	Keyboard annunciator (ANCTRL)	(ANFLAG)
none	000000	000000	000000
←	001E28	n/a	n/a
AC	101E21	101E21	7E6F21
USER	101E22	n/a	n/a
RAD	101E24	n/a	n/a
BAT	301E21	n/a	n/a
0	C43E28	C42E28	9E6F21
1	D43E21	D43E21	9E6F22
2	D43E22	D43E22	9E6F24
3	D43E24	D43E24	9E6F28
4	D43E28	D43E28	AE6F21
((•))	E43E28	E43E28	7E6F28
→	F43E21	n/a	n/a
PRGM	F43E22	n/a	n/a
SUSP	F43E24	n/a	n/a
CALC	F43E28	n/a	n/a

Display Control Strings

Three strings are used by the debugger to control the HP-IL display device before displaying a register. The strings are similar to the ENDLINE string in BASIC, except that the debugger sends these strings *before* displaying a register, not after.

Appendix C, "Addresses of Entry Points," gives directions for finding the locations of these control strings.

You can modify the contents of these strings to customize the debugger output. They are designed for controlling output devices, and not for labelling or printing extra characters. Normally you cannot specify printable characters in these strings, but only *control characters* — that is, characters such as CR or LF, or escape sequences. If you include printable characters they must be followed by a CR; otherwise the cursor action will not match the correct editable positions. Also, the debugger ignores null characters (CHR$(0)), even if included in an escape sequence.

The NewLin string is used when a register is about to be sent to the display device in view mode. Its default contents are: CR/LF and six "FF" terminators. You can specify any eight control characters for this string; if less than eight are used, the string should end in an "FF" terminator.

The EdtLin string is used when a register is about to be sent to the display device in edit mode. Its default contents are: CR and seven "FF" terminators. You can specify any eight control characters for this string; if less than eight are used, the string should end in an "FF" terminator.

For example, if your HP-IL display device is a printer, you might want to change the EdtLin string to a CR/LF. Otherwise, when in edit mode the printer would overstrike the line as new characters are sent out.

The VdoLin string is used when a register is about to be sent to the display device in view mode and the "V" option is set. Its purpose is to "roll" up the display screen without disturbing the 80-column register template. The VdoLin location actually contains a five-nibble address of the desired string; if this field is 00000, the default string V80Lin (in DBGLEX3A) is used. V80Lin contains:

1. Escape > (to make the template-building visible on an HP-IL video interface);
2. Escape H (home cursor);
3. 10 line feeds;
4. Escape M (delete line);
5. 12 line feeds;
6. CR/LF and an "FF" terminator.

The V80Lin string is followed by 16 unused bytes; if DBGLEX3A is in RAM, you can expand the string up to 46 control characters and an "FF" terminator. Alternatively, you can find an available area of RAM to store the string and place its address in the VdoLin field. Since the debugger's display buffer accommodates only 32 characters, if you choose to output printable characters you cannot include more than 32.

For example, if your device has 16-line by 80-column capability, you would want to delete eight of the line feeds from item 5 in the V80Lin string. If the device is a printer, you could also delete the line feeds to save paper.

Modifying the KEYBOARD IS Mapping

The debugger allows the KEYBOARD IS device to direct the system by mapping non-ASCII keystrokes to control characters. At the location CTRLTB in DBGLEX2A resides a table of 26 entries corresponding to the control codes CTRL A through CTRL Z. Each two-nibble entry is a keycode for mapping the corresponding control sequence to a keystroke recognized by the debugger. For example, the eighteenth entry is "kcRUN" (0F hex) to map CTRL R to the [RUN] key. An "FF" entry indicates no corresponding key mapping.

Section 8, "Using HP-IL," contains a table of the default key mappings for KEYBOARD IS. If the file DBGLEX2A is in RAM, you can modify or add to the entries in the CTRLTB table to map other keystrokes into the debugger.

Note that mapping the [ATTN] key in the CTRLTB table is not a simple matter, since some actions performed in the interrupt routine must be duplicated. These include setting status bit S12 and writing an "F" to location "DATNFL" ("debugger ATTN flag"). Without these actions the mapped [ATTN] key will not work properly. However, the KEYBOARD LEX file has provisions for assigning the [ATTN] key to an ESCAPE code.

Customizing REENTR

The assembly language reentry point, REENTR, uses the nibble at location "EmRntr" in DBGLEX1A to set up entry conditions for the emulator. You can set or clear bits in this nibble before you call REENTR for the following effects:

- Bit 3 (1xxx): when set, a break point on the first instruction will be ignored.
- Bit 2 (x1xx): when set, the single-step flag will be set, causing a break after the first instruction.
- Bit 1 (xx1x): not used.
- Bit 0 (xxx1): when set, the ((•)) annunciator will be turned on, indicating that keystrokes are sent to the mainframe key buffer.

After reading and testing this nibble, the emulator clears it. If you are using the EmRntr features, your routines need to set the nibble each time they call REENTR.

Code Modification

The modular design of the debugger allows relatively easy customization. There are built-in features to let applications use the display routines, intercept keys, or extend the instruction set of the emulator. This manual does not describe interfaces to debugger routines, but advanced users — with the debugging and disassembling capabilities of the system — can use the list of entry points in appendix C to make incremental enhancements to the software.

There are several empty areas in the code space which can be used for "GOTO" links or bug fixes. The file DBGLEX3A takes less than 3K bytes, leaving a large area in an IRAM for enhancements. Refer to appendix C for locations of these and other important entry points.

Appendix A

Warranty and Service Information

Limited One-Year Warranty

What We Will Do

The HP 82478A Debugger medium is warranted by Hewlett-Packard against defects in materials and workmanship affecting electronic and mechanical performance, but not software content, for one year from the date of original purchase. If you sell your unit or give it as a gift, the warranty is transferred to the new owner and remains in effect for the original one-year period. During the warranty period, we will repair, or, at our option, replace at no charge a product that proves to be defective, provided you return the product, shipping prepaid, to a Hewlett-Packard service center.

What Is Not Covered

This warranty does not apply if the product has been damaged by accident or misuse or as the result of service or modification by other than an authorized Hewlett-Packard service center.

No other express warranty is given. The repair or replacement of a product is your exclusive remedy. **ANY OTHER IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS IS LIMITED TO THE ONE-YEAR DURATION OF THIS WRITTEN WARRANTY.** Some states, provinces, or countries do not allow limitations on how long an implied warranty lasts, so the above limitation may not apply to you. **IN NO EVENT SHALL HEWLETT-PACKARD BE LIABLE FOR CONSEQUENTIAL DAMAGES**. Some states, provinces, or countries do not allow the exclusion or limitation of incidental or consequential damages, so the above limitation or exclusion may not apply to you.

This warranty gives you specific legal rights, and you may also have other rights which vary from state to state, province to province, or country to country.

Warranty for Consumer Transactions in the United Kingdom

This warranty shall not apply to consumer transactions and shall not affect the statutory rights of a consumer. In relation to such transactions, the rights and obligations of Seller and Buyer shall be determined by statute.

Obligation to Make Changes

Products are sold on the basis of specifications applicable at the time of manufacture. Hewlett-Packard shall have no obligation to modify or update products once sold.

Warranty Information

If you have any questions concerning this warranty, please contact an authorized Hewlett-Packard dealer or a Hewlett-Packard sales and service office. Should you be unable to contact them, please contact:

- In the United States:

Hewlett-Packard
Personal Computer Group
Customer Support
11000 Wolfe Road
Cupertino, CA 95014

Toll-Free Number: (800) FOR-HPPC (800 367-4772)

- In Europe:

Hewlett-Packard S.A.
150, route du Nant-d'Avril
P.O. Box CH-1217 Meyrin 2
Geneva
Switzerland
Telephone: (022) 83 81 11

Note: Do *not* send units to this address for repair.

- In other countries:

Hewlett-Packard Intercontinental
3495 Deer Creek Rd.
Palo Alto, California 94304
U.S.A.
Telephone: (415) 857-1501

Note: Do *not* send units to this address for repair.

Service

Hewlett-Packard maintains service centers in most major countries throughout the world. You may have your unit repaired at a Hewlett-Packard service center any time it needs service, whether the unit is under warranty or not. There is a charge for repairs after the one-year warranty period.

Hewlett-Packard products are normally repaired and reshipped within five (5) working days of receipt at any service center. This is an average time and could vary depending upon the time of year and the work load at the service center. The total time you are without your unit will depend largely on the shipping time.

Obtaining Repair Service in the United States

The Hewlett-Packard United States Service Center for battery-powered computational products is located in Corvallis, Oregon:

Hewlett-Packard Company
Service Department

P.O. Box 999
Corvallis, Oregon 97339, U.S.A.
or
1030 N.E. Circle Blvd.
Corvallis, Oregon 97330, U.S.A.

Telephone: (503) 757-2002

Obtaining Repair Service in Europe

Service centers are maintained at the following locations. For countries not listed, contact the dealer or sales office where you purchased your unit.

AUSTRIA
HEWLETT-PACKARD Ges.m.b.H.
Kleinrechner-Service
Wagramerstrasse-Lieblgasse 1
A-1220 Wien (Vienna)
Telephone: (0222) 23 65 11

BELGIUM
HEWLETT-PACKARD BELGIUM SA/NV
Woluwedal 100
B-1200 Brussels
Telephone: (02) 762 32 00

DENMARK
HEWLETT-PACKARD A/S
Datavej 52
DK-3460 Birkerod (Copenhagen)
Telephone: (02) 81 66 40

EASTERN EUROPE
Refer to the address listed under Austria.

FINLAND
HEWLETT-PACKARD OY
Revontulentie 7
SF-02100 Espoo 10 (Helsinki)
Telephone: (90) 455 02 11

FRANCE
HEWLETT-PACKARD FRANCE
Division Informatique Personnelle
S.A.V. Calculateurs de Poche
F-91947 Les Ulis Cedex
Telephone: (6) 907 78 25

GERMANY
HEWLETT-PACKARD GmbH
Kleinrechner-Service
Vertriebszentrale
Berner Strasse 117
Postfach 560 140
D-6000 Frankfurt 56
Telephone: (611) 50041

ITALY
HEWLETT-PACKARD ITALIANA S.P.A.
Casella postale 3645 (Milano)
Via G. Di Vittorio, 9
I-20063 Cernusco Sul Naviglio (Milan)
Telephone: (2) 90 36 91

NETHERLANDS
HEWLETT-PACKARD NEDERLAND B.V.
Van Heuven Goedhartiaan 121
NL-1181 KK Amstelveen (Amsterdam)
P.O. Box 667
Telephone: (020) 472021

NORWAY
HEWLETT-PACKARD NORGE A/S
P.O. Box 34
Oesterndalen 18
N-1345 Oesteras (Oslo)
Telephone: (2) 17 11 80

SPAIN
HEWLETT-PACKARD ESPANOLA S.A.
Calle Jerez 3
E-Madrid 16
Telephone: (1) 458 2600

SWEDEN
HEWLETT-PACKARD SVERIGE AB
Skalholtsgatan 9, Kista
Box 19
S-163 93 Spanga (Stockholm)
Telephone: (08) 750 2000

SWITZERLAND
HEWLETT-PACKARD (SCHWEIZ) AG
Kleinrechner-Service
Allmend 2
CH-8967 Widen
Telephone: (057) 31 21 11

UNITED KINGDOM
HEWLETT-PACKARD Ltd
King Street Lane
GB-Winnersh, Wokingham
Berkshire RG11 5AR
Telephone: (0734) 784 774

International Service Information

Not all Hewlett-Packard service centers offer service for all models of HP products. However, if you bought your product from an authorized Hewlett-Packard dealer, you can be sure that service is available in the country where you bought it.

If you happen to be outside the country where you bought your unit, you can contact the local Hewlett-Packard service center to see if service is available for it. If service is unavailable, please ship the unit to the address listed above under Obtaining Repair Service in the United States. A list of service centers for other countries can be obtained by writing to that address. All shipping, reimportation arrangements, and customs costs are your responsibility.

Service Repair Charge

There is a standard repair charge for out-of-warranty repairs. The repair charges include all labor and materials. In the United states, the full charge is subject to the customer's local sales tax. In

European countries, the full charge is subject to Value Added Tax (VAT) and similar taxes wherever applicable. All such taxes will appear as separate items on invoiced amounts.

Computer products damaged by accident or misuse are not covered by the fixed repair charges. In these situations, repair charges will be individually determined based on time and materials.

Service Warranty

Any out-of-warranty repairs are warranted against defects in materials and workmanship for a period of 90 days from date of service.

Shipping Instructions

Should your unit require service, return it with the following items:

- A completed Service Card, including a description of the problem.
- A sales receipt or other proof of purchase date if the one-year warranty has not expired.

The product, the Service Card, a brief description of the problem, and (if required) the proof of purchase date should be packaged in adequate protective packaging to prevent in-transit damage. Such damage is not covered by the one-year limited warranty; Hewlett-Packard suggests that you insure the shipment to the service center. The packaged unit should be shipped to the nearest Hewlett-Packard designated collection point or service center. Contact your dealer for assistance. (If you are not in the country where you originally purchased the unit, refer to "International Service Information" above.)

Whether the unit is under warranty or not, it is your responsibility to pay shipping charges for delivery to the Hewlett-Packard service center.

After warranty repairs are completed, the service center returns the unit with postage prepaid. On out-of-warranty repairs in the United States and some other countries, the unit is returned C.O.D. (covering shipping costs and the service charge).

Further Information

Circuitry and designs are proprietary to Hewlett-Packard, and service manuals are not available to customers.

Should other problems or questions arise regarding repairs, please call your nearest Hewlett-Packard service center.

When You Need Help

Hewlett-Packard is committed to providing after-sale support to its customers. To this end, our customer support department has established phone numbers that you can call if you have questions about this product.

Product Information. For information about Hewlett-Packard dealers, products, and prices, call the toll-free number below:

(800) FOR-HPPC
(800) 367-4772

Technical Assistance. For technical assistance with your product, support specialists are available to assist you Monday through Friday, from 8:00 a.m. to 3:00 p.m. Pacific Time. Call:

(503) 757-2004

For either product information or technical assistance, you can also write to:

Hewlett-Packard
Handheld Computer and Calculator Operation
Calculator Technical Support
1000 N.E. Circle Blvd.
Corvallis, OR 97330

Appendix B

BASIC Keywords

The DBGMAINA file provides three BASIC keywords to access and set up the debugger.

DBGADDR$

DBGADDR$(*file number*)

The *file number* is 1, 2, or 3, corresponding to DBGLEX1A, DBGLEX2A, or DBGLEX3A. This function returns the main entry address of the active DBGLEX file; this is not the same address as returned by ADDR$.

DBGADDR$ returns 00000 if the corresponding DBGLEX file is not found or if it resides in an unacceptable device or address. The DBGLEX files are found by internal coding, not by file name. You can rename the DBGLEX files as desired, and DBGADDR$ will still return the main entry addresses.

Acceptable devices and addresses for DBGLEX files

File	Acceptable	Unacceptable
DBGLEX1A	Must reside in RAM or IRAM, at an address ending in "008".	ROM, PROM, any address not ending in "008".
DBGLEX2A	RAM, IRAM, ROM, PROM.	None.
DBGLEX3A	RAM, IRAM, ROM, PROM.	None.

DDISPLAY IS

DDISPLAY IS *device specifier*

This statement specifies an HP-IL display device for debugger output. Syntax for this statement is identical to the DISPLAY IS statement in the HP-IL ROM. Refer to page 75 of the *HP-IL Interface Owner's Manual* for details on device specifiers.

DDISPLAY IS accepts the following device specifiers:

- Accessory type. Example: DDISPLAY IS %48:2
- Device word. Example: DDISPLAY IS RS232
- HPIL address. Example: DDISPLAY IS 4
- Assign code. Example: DDISPLAY IS ":TV" (after executing an ASSIGN IO statement which assigns ":TV" to a device).

DDISPLAY IS does not accept the following device specifiers:

- Device type. Example: DDISPLAY IS :HP82905B
  Causes: DBG ERR:Illegal Device Spec
- Volume label. Example: DDISPLAY IS .STORE
  Causes: DBG ERR:Illegal Device Spec

In addition, DDISPLAY IS * not only cancels the display assignment, but clears the "H" option in the debugger and zeroes the display handler address.

DEBUG

DEBUG
DEBUG *

The **DEBUG** statement activates the debugger, sending control to the keyboard. The **DEBUG** * statement is similar, but it begins emulation immediately without requiring keyboard direction.

Files DBGLEX1A and DBGLEX2A must be in HP-71 memory in acceptable memory devices to activate the debugger. File DBGLEX3A is optional. (Refer to the DBGADDR$ keyword for a list of acceptable devices.)

Appendix C

Addresses of Entry Points

The following list of entry points is for the "DBG:A" version of the debugger. The offsets should be used with the DBGADDR$ function, which is included in the DBGMAINA file.

DBGADDR$ is used to find the address of the active DBGLEX files. Its syntax is as follows:

DBGADDR$(*file number*)

where *file number* is 1, 2, or 3, corresponding to DBGLEX1A, DBGLEX2A, or DBGLEX3A. If a file is not found (or is located in an unacceptable device) DBGADDR$ returns "00000". The addresses returned are those of main entry points in each file.

For instance, after you copy DBGLEX1A into IRAM, DBGADDR$(1) might return "68055". Use this address to locate entry points in file DBGLEX1A. For example, to locate the absolute address of REENTR, evaluate the following expression:

DTH$(HTD(DBGADDR$(1)) + HTD("000A8"))

When working in the debugger, you can easily determine absolute addresses of the DBGLEX entry points by using the three "Wndw + DBGADDR$" commands in the extended command menu. First, select a register in the Window stack, then edit and place in it one of the offsets from the following tables. Enter the command menu with g[CMDS] and select the appropriate "Wndw + DBGADDR$" command. Press [ENDLINE] and the window address will be adjusted to the absolute address of the entry point.

The debugger affords easy access to memory, anywhere in the HP-71. Whether you use the BASIC command "POKE" or the memory edit feature of the debugger, you should not change the contents of memory without double-checking your procedure. A slight error in address or nibble values can result in a Memory Lost or the corruption of files or data. Be sure you understand the requirements of HP-71 file structure and memory management before you modify RAM contents or the debugger code.

Table of offsets for DBGLEX1A:

- GN1JMP — 00000. General-purpose jump point for DBGLEX1A.
- REENTR — 000A8. Entry point for reentering the debugger from assembly language.
- INTVCT — 000F2. Interrupt vector address.
- SpaceA — 00775. 20 free nibbles, available for bug fixes or code enhancements.
- BegRAM — 00789. Beginning of the debugger RAM map.
- RCVRA — 00789. Storage of A register for RECOVER; all CPU registers are stored in this area for the RECOVER sequence.

- V80:PC — 007AB. Output buffer for 80-column video interface.
- MANHOL — 0085D. Control nibble for key buffers. If bit 0=1, then keys are put into the mainframe KEYBUF. If bit 0=0, keys are put into DKYBUF.
- ANCTRL — 0086E. [ON]g annunciator control. The first five nibbles are the address of the annunciator, the next nibble is the mask to set it.
- ANFLAG — 00874. [ON]g annunciator flag control. The first five nibbles are the address of the system flag, the next nibble is the mask to set it. (Refer to page 201 of the *HP-71 Owner's Manual* for a list of flags which control annunciators.)
- DATNFL — 0087B. Debugger's [ATTN] key flag, used similarly to ATNFLG in the mainframe.
- DKYPTR — 0087C. Debugger's key buffer pointer; used similarly to KEYPTR in the mainframe.
- DKYBUF — 0087D. Debugger's key buffer; 15 bytes used similarly to KEYBUF in the mainframe.
- Reg:A — 008AB. Start of register storage for CPU emulation.
- EmRntr — 00958. Control for reentering the emulator (such as from a configuration jump). If bit 3 is set, a break point on the first instruction will be ignored. If bit 2 is set, the single-step flag will be set. If bit 0 is set, the alarm annunciator ((•)) will be turned on. (Bit 1 is not used.) After reading and testing this nibble, the emulator clears it.
- ChkSm1 — 00AA7. Storage location for the checksum byte for DBGLEX1A. Bytes for DBGLEX2A and DBGLEX3A follow immediately.
- USCTRL — 00AAD. User stack annunciator control. The first five nibbles are the address of the annunciator, the next nibble is the mask to set it.
- DB3Kbd — 00AEF. Nibble in RAM which indicates if an application in DBGLEX3A needs to intercept keys from DBGLEX2A. (Bit 0 is used by the extended command menu; the other three bits are available for enhancements.)
- ChrBuf — 00AF3. Buffer for building debugger displays.
- Dspchx — 00B3B. Storage of the address of the HPIL display code. Normally, this would be the same address as found in the mainframe RAM location DSPCHX.
- NewLin — 00B8F. String of eight characters to send out when displaying a new line in view mode. Default: D0A0FFFFFFFFFFFF (CR,LF and six terminators). These eight bytes can be changed to any desired characters, much like the ENDLINE statement in BASIC. If less than eight are used, they should be terminated with an "FF" byte.
- EdtLin — 00BA1. String of eight characters to send out when rebuilding a display during editing. Default: D0FFFFFFFFFFFFFF (CR and seven terminators). These eight bytes can be changed to any desired characters, much like the ENDLINE statement in BASIC. If less than eight are used, they should be terminated with an "FF" byte. If the display device is a printer, you will probably want to change the first two bytes to a CR,LF.
- VdoLin — 00BB3. Five nibble storage of address for display string to maintain the 80-column template. If this field is 00000, the default string V80Lin in DBGLEX3A is used. To change the output string, you can choose an unused area of RAM to fill in a different string, and fill in the address in VdoLin. Or you can modify the V80Lin string if DBGLEX3A is in RAM.
- RegXQ+ — 00BE8. Two nibbles following the XQT register, available for editing for extended opcodes ("LC(15)" and "LC(16)").

- HPILfl — 00BEA. Active HPIL device flag. When this flag is "0", the mainframe HPIL control locations are set up for the mainframe display device. When this flag is "F", the mainframe HPIL control locations are set up for the debugger display device. If your display devices become swapped for some reason (this may happen if HPIL issues an error while in the debugger, for instance), you can change the value of this nibble to reswap the devices.
- mbox^ — 00BEB. Debugger's equivalent to MBOX^ field. Along with the following three fields (loopst, is-dsp, and dspset), these are swapped to change display devices between the mainframe and the debugger.
- loopst — 00BEE. Debugger's equivalent to LOOPST nibble.
- is-dsp — 00BEF. Debugger's equivalent to IS-DSP field.
- dspset — 00BF6. Debugger's equivalent to DSPSET nibble.
- MFDRAM — 00BF7. Location for storage of the mainframe display buffer, 224 nibbles, starting with DSPSTA+3.
- Annad1 — 00CD7. Location for storage of 96 nibbles from display driver at address 2E100. Following this are two more 96 nibble blocks for display drivers as 2E200 and 2E300.
- RUN19 — 0114C. Entry point for emulator, to set up registers and controls.
- NSSX-R — 011F3. Main loop address for the emulator.
- BpVol — 01F6F. Debugger beep volume control. This nibble is an "8" for low beep volume; change to a "4" for loud volume. It controls all beeps issued by the debugger.
- DBG1EN — 01F92. End of DBGLEX1A. If the file is the last in an IRAM, there are 23 free nibbles before the end of the IRAM, available for bug fixes or code enhancements. (The file length in the DBGLEX1A header would need to be adjusted if code were added here.)

Table of offsets for DBGLEX2A:

- GN2JMP — 00000. General-purpose jump point for DBGLEX2A.
- SpaceB — 00022. 18 free nibbles, available for bug fixes or code enhancements.
- DSPREG — 00034. Routine to display a debugger register.
- SpaceC — 00AE3. 18 free nibbles, available for bug fixes or code enhancements.
- POPCHK — 00AF5. Wait 1½ seconds for key, but don't pop it from key buffer.
- DPOPBF — 00B28. Routine to pop a key from the debugger's key buffer.
- CTRLTB — 00C2C. Table of control codes for mapping to debugger keys. 26 entries correspond with the keystrokes CTRL A through CTRL Z.
- SLWTKY — 00E24. Idle loop, "go to sleep, wait for a key."
- SpaceD — 01457. 18 free nibbles, available for bug fixes or code enhancements.
- SpaceE — 01B83. 18 free nibbles, available for bug fixes or code enhancements.
- CBf2DS — 01B98. Display the debugger character buffer, ChrBuf.
- DspCHC — 01C9D. Display a character, byte in C(B).
- BldDsp — 01DD8. Routine which builds the debugger's display buffer in the LCD.

- DspUpd — 01DF0. Update the LCD, using the debugger's display buffer.
- DspRst — 01EC2. Reset the debugger's display.
- DBG2EN — 01F90. End of DBGLEX2A. If the file is the last in an IRAM, there are 25 free nibbles before the end of the IRAM, available for bug fixes or code enhancements. (The file length in the DBGLEX2A header would need to be adjusted if code were added here.)

Table of offsets for DBGLEX3A:

- GN3JMP — 00000. General-purpose jump point for DBGLEX3A.
- SpaceF — 00021. 27 free nibbles, available for bug fixes or code enhancements. This space immediately follows the jump table at GN3JMP, so that more entries can be included for future enhancements.
- BIASap — 0004B. Routine which applies the BIAS to an address.
- V80COL — 00137. 80-column video interface output routines.
- V80Lin — 00159. The string used by default for the 80-column video interface. This string is sent out before displaying a register when the "V" option is set.

  This default string consists of an Escape >, Escape H, 10 LFs (line feeds), an Escape M, 12 LFs, a CR/LF, an "FF" terminator, and 16 unused bytes. There are 47 bytes available here; if DBGLEX3A is in RAM, you can change the contents of this default VdoLin string as desired. (For an alternative to changing V80Lin, refer to the VdoLin entry point in DBGLEX1A.)
- JpHPIL — 00619. Jump into HPIL module, using the Dspchx address and an offset to the desired routine.
- SpaceG — 00741. 24 free nibbles, available for bug fixes or code enhancements.
- DCMPL — 0079E. Decompile routine.
- gCMD — 01155. Extended command menu routines.
- SpaceH — 0129D. 52 free nibbles, available for bug fixes or code enhancements. This space is in the table of command menus, so that more can be added for future enhancements.
- SpaceI — 0131A. 18 free nibbles, available for bug fixes or code enhancements. This space immediately precedes the DB3Key entry point, and is available for enhancements which might intercept keys.
- DB3Key — 0132C. Routine in DBGLEX3A which intercepts keys. (For example, used by the extended command stack to interpret the up- and down-arrow keys, as well as [ATTN] and [ENDLINE].) If a routine in DBGLEX3A does not use the key, it is handled by the main loop at SLWTKY in DBGLEX2A.
- DBG3EN — 01645. End of DBGLEX3A. If the file is the last in an IRAM, there are 2404 free nibbles before the end of the IRAM, available for bug fixes or code enhancements. (The file length in the DBGLEX3A header would need to be adjusted if code were added here.)

HEWLETT PACKARD
Portable Computer Division
1000 N.E. Circle Blvd., Corvallis, OR 97330, U.S.A.

European Headquarters
150, Route Du Nant-D'Avril
P.O. Box, CH-1217 Meyrin 2
Geneva-Switzerland

HP-United Kingdom
(Pinewood)
GB-Nine Mile Ride, Wokingham
Berkshire RG11 3LL

82478-90001 English

Printed in Singapore 7/86