DebugZ

A Symbolic Debugger for Z80/H64180

Release 1.2

April, 1989

Created by
MICROCode Consulting
www.microcodeconsulting.com

CP/M 2.2 is a registered trademark of Digital Research, Incorporated. Z-80 is a registered trademark of Zilog, Incorporated. NSC-800 is a trademark of National Semiconductor Corporation. WordStar is a trademark of MicroPro International.

This document describes DebugZ, a full-screen capable symbolic debugger for Z80/H64180 microprocessors running CP/M or QP/M. Included with this package is binary executable code. The documentation and binary executable code are hereafter designated as Contents.

MICROCode Consulting hereby grants that Contents is released for unlimited distribution. Contents may be given , transmitted, reproduced and utilized without charge excepting the actual cost of reproduction. Contents may not be included in any commercial or for-profit package without the express written permission of MICROCode Consulting.

DISCLAIMER

MICROCode Consulting makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, MICROCode Consulting reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of MICROCode Consulting to notify any person of such revision or changes.

TABLE OF CONTENTS

A. Introduction

DebugZ is a powerful symbolic debugger for Z80 and H64180 (Z180) microprocessors running under the CP/M or QP/M operating system. Here it is, 1989, with a bucket of symbolic debuggers available and now there is yet another one. Why?

First , writing a debugger was not on our list. Off the shelf debuggers were used starting eight years ago with DDT, ZSID , and finally DCON. DCON is not true Z80 (using TDL mnenomics), but it was in the mold of DDT/ZSID and served well until a couple years ago. Then, its propensity for crashing upon exit and trashing the DOS jump vector when RSXes are present became a problem.

The research to find a suitable replacement ensued: ZDT, Debug17 , DDTZ , Z8E , 18E , DSD and a few others. Criteria for the new debugger included:

- operation upward compatible with DDT/ZSID/DCON
- on-line help file
- both normal and full-screen trace mode
- instruction look-ahead
- Zilog Z80 mnenomics
- Hitachi 64180 (Z180) instructions and support
- low cost

Although each debugger examined had some of the features desired, only DSD and Z8E (18E) had a full-screen debug mode. DSD is pricey and Z8E has command set unique to itself. No debugger had a nice single-keystroke trace mode with instruction look-ahead or an on-line help file. Thus, a new debugger program was born.

B. Requirements and Restrictions

As with any product, there are some requirements and restrictions in using DebugZ. All the code is written for Z-80 or compatible computers on the CP/M or QP/M operating system. Requirements are pretty basic:

- minimum of CP/M 2.x or QP/M or compatibles
- Z80, H64180, Z180, or NSC-800 microprocessor
- 12k of disk storage for the main program, 38k if one also wishes to have an on-line help file

There is only one restriction to using DebugZ: If you wish to use the full-screen debugger portion, your video terminal must be capable of full-screen operation.

C. Installing DEBUGZ.COM

Although you can run DebugZ "right out of the box" , you might wish to customize it for your personal use. You will need the General Terminal Facilities program, also by MICROCode Consulting , in order to install the terminal. (This program is now included in **DEBUGZ.LBR**.)

In order to change the default values of DebugZ, you will need both **DBGINST.COM** and **DEBUGZ.COM** on the same drive and user area. Execute the installation program

```
DBGINST
```

which will bring up the following menu:

```
          DebugZ Values and Options

     Restart <V>ector at:     0038H  (8080 equiv. of RST 7)
     Computer <S>peed in MHz: 4.0
     Strip high-bit of <A>SCII chars during dump: No
     Display <Z>80 secondary registers: No
     <T>race of program execution < 0100H (DOS): No
     Display instructions in <U>pper case: Yes

     Terminal t<Y>pe:  <<< not installed >>>

     <R>estore starting values
     <W>rite changes and exit
     e<X>it without writing changes
```

and are individually described here.

Restart <V>ector: Normally, the restart vector is set to 0038H (RST 7) for debuggers. However, if you need to use a different vector for some reason, you can customize it for any of the eight possible restart vectors (RST 0 through RST 7).

Computer <S>peed: When running the debugger in single-step trace mode, DebugZ uses a delay routine to determine the stepping speed. This value is used to obtain the proper step rate. If your system is single-step tracing too fast, raise the <S>peed; conversely, if too slow, lower it.

Strip <A>SCII high bit: Normally, only true ASCII values are shown on the DUMP display as characters. Thus, any bytes with the high-bit set are displayed as periods ("."). However, if you wish to see the ASCII equivalent of all characters, after any high-bit is stripped, then set this to "Yes".

Display <Z>80 secondary_registers: DebugZ always displays the primary registers: AF, BC, DE, HL, SP, and PC. Since Z80 registers are not always used by a program (and doing so requires an extra line of display), DebugZ defaults to not displaying them. If you wish to normally view the Z80 registers (AF', BC', DE', IX, and IY), then set this to "Yes". NOTE: This value can be changed while DebugZ is running via an "E"xtended command.

Trace through DOS: Debuggers usually disable program trace when the PC actually enters DOS. DebugZ offers the option to enable this by default. NOTE: This value can be changed while DebugZ is running via an "E"xtended command.

Display instructions <U>pper: **B**y default, DebugZ displays instructions in upper-case characters because that is the way the author likes them. You may feel differently.

Terminal t<Y>pe: DebugZ comes uninstalled for any specific terminal. If you wish to use the full-screen capabilities of DebugZ, you must install a terminal definition. Note that this terminal must have insert line and delete line capability (which is most machines).

To install a new terminal (or change an old one), you must have the file **QTERM.DAT** on your system. This file is a terminal definition file produced by **SETQTERM.COM** in the QTERM General Terminal Facilities library. Consult that documentation for use of its programs.

<R>estore starting values: If you do not like the changes you made, this option restores the starting values without leaving DBGINST.

<W>rite_changes_and_exit: If the changes made are correct, this option modifies the DebugZ program and exits.

e<X>it_without_writing_changes: If the changes made are incorrect or you wish to abort, this option exits without modifying DebugZ.

D. Using DEBUGZ.COM

In this section, execution of DebugZ is described. Numeric values that it accepts are also presented.

D-1. Executing DebugZ

DebugZ can be executed with or without command line parameters. All values in "[]" are optional, with an asterick ("*") being the place holder for "program" when only symbols are loaded.

```
DEBUGZ  [program  [symbol]] [=addr]
```

where "`program`" is an optional program file (usually filename.COM); "`symbol`" is an optional symbol file (usually filename.SYM) and "`addr`" is an optional base address in hexadecimal to load DebugZ. A base address must be preceded by an equal sign ("="); by default, DebugZ will locate itself at the highest location possible (just below DOS) which rarely needs to be changed.

D-2. Values

Literal Numbers: Default are hexadecimal values; any numbers from 0-9 and letters A-F are accepted. (Ex: 0F0D 7A)

Hexadecimal Numbers: Same as literal numbers; also, any number preceded by "\" and "H" ("\H"). (Ex: `\HE77`)

Decimal Numbers: Decimal numbers are preceded by either a "#" sign or "\" and "D" ("\D"); digits from 0-9 are accepted. (Ex: `#9178 #31419 \D3712`)

Binary Numbers: Binary numbers are preceded by a "\" and "B" ("\B"); digits from 0-1 are accepted. (Ex: `\B11100110`)

Characters: Any set of ASCII characters is accepted within paired quotes, either ' or " (must be matching pair). The rightmost char becomes the least significant; a one character string has a zero high-order byte. Case is not translated within quotes. (Ex: "cd" 'y' 'T"' "'7"')

Stack values: Stack values are obtained with the dollar sign ("$") symbol. A sequence of n "$" obtains the n^{th} stacked value in the current program. (Ex: $ = top of stack, $$$ = 3rd value on stack)

D-3. Symbol References

A symbol is a sequence of ASCII characters ("sym") as read in by DebugZ. Symbols are located in a .SYM symbol file.

Valid symbol references are:

`.sym` 16-bit value of the symbol "sym" (Ex: `.MYSYM`)
`@sym` 16-bit value POINTED TO by symbol "sym" (Ex: `@MYSYM`)
`=sym` 8-bit value POINTED TO by symbol "sym" (Ex: `=MYSYM`)

D-4. Expressions

An expression is any collection of numbers, characters, values or symbol references separated by binary operators or the three unary operators. Note that overflow is NOT detected.

Examples:
```
.MYSYM+#145*3
@(.MYSYM+14)/#71-^B0110
!+#4124
```

Note that indirect references ("@" and "=") are valid using any expression in parenthesis. The last example surfaces a third reference, "`!`", which uses the result from the *last* expression. Note that a leading "+" is implicitly preceded by a "`!`".

D-5. Operators

DebugZ supports a wide range of unary and binary operators. The list of operators by precedence order and name are:

Group A:	~ (bitwise not)	() (parenthesis)	
Group B:	@ (16-bit value)	= (8-bit value)	
Group C:	* (`multiply`)	`/ (divide)`	`% (modulo)`
Group D:	+ (addition)	- (subtract)	
Group E:	& (bit AND)	| (bit OR)	^ (bit XOR)

Groups A and B are unary operators. "@" and "=" *must* be followed by either a symbol name or expression inside parenthesis. Groups C, D and E are binary operators.

E. DebugZ Commands

DebugZ commands are detailed in this section. First, a brief summary is given, followed by the command syntax. Each command starts with a single letter and is followed by one or more parameters separated by commas. Some commands allow a minus sign "-" to precede the single command letter. Unless otherwise specified, a parameter can be any expression including symbols and operators. The full 16-bit value is used by DebugZ with few exceptions; these exceptions are noted in the text.

Optional parameters are included in square brackets "[]". If optional parameters are nested (brackets inside of brackets), then the outer parameter must be specified in order for DebugZ to recognize the inner parameter.

Finally, a detailed description for the command is given followed by one or two examples. In any example, user input is underlined; DebugZ output is shown in normal text. Useful comments are preceded by a semicolon ";".

E-1. Assemble

Summary: Enter assembly language instructions

Syntax: A[s]

Description: The ASSEMBLE command allows one to enter one or more Z80 or H64180 assembler instructions that are then stored consecutively into memory. A specific load address "s" can be specified; in the absence of an address "s", the default is to use the address after the last assembly, list or trace command.

The user is prompted by the assembly address followed by a colon. Commands are accepted one line at a time. Entering a blank line or "." terminates assembly.

Example:

```
#A100                   ;assemble at address 0100H

0100: LD HL,#8190
0103: OTIR
0105: .                 ;period to quit

#A

0105: RET
0106:                   ;empty line to quit
```

E-2. Call

Summary: Excecute a subroutine with optional parameters

Syntax: Cs[,b[,d]]

Description: The CALL command executes a subroutine at location "s" with optional parameters "b" and "d". If given, the register pair BC is set to "b"; likewise, register pair DE is set to "d". The default value for BC and DE is zero (0). Control returns to the DebugZ monitor upon normal exit of the subroutine.

NOTE: This command is *not* considered part of the program under test. The CPU state of the program is not altered. All registers are reset back to their original program value upon subroutine completion.

Example:

```
#C.SUB1                  ;call routine SUB1
#Cea03,#500,\B1101       ;call EA03H with BC=#500 and DE=\B1101
```

E-3. Display memory

Summary: Display a hexadecimal/ASCII dump of memory

Syntax: [-]D[W][b][,e]

Description: The DISPLAY command dumps memory with the hexadecimal values on the left part of the display and their matching ASCII characters on the right. Normally, 12 lines (192 bytes) of memory are displayed starting immediately after the last display address shown. A different or specific beginning address is specified by supplying the "b" parameter. Similarly, the dump will terminate at the ending address "e", if given, overriding the normal 12 line dump limit. However, DebugZ automatically pauses (paginates) after each set of 16 lines (256 bytes) for the user to catch up.

If the user wishes to display 16-bit words rather than 8-bit bytes, then the "W" should appear immediately after the "D". Finally, pagination can be suppressed by preceding the "D" with a minus sign "-".

Example:

```
#D100                   ;dump at address 0100 for 12 lines
#D,200                  ;continue displaying up to 0200
#-DW.SUB1,.SUB1+80      ;dump word values from address SUB1
                        ;through SUB1+80 without pagination
```

E-4. Extended Commands

Summary: Change the display environment and trace mode.

Syntax: E
ERtype
[-]ETRACE

Description: Extended commands direct the operation of DebugZ as follows:

```
ERINTEL     ; Display (Intel) 8080 registers only.
ER8080      ; Display (Intel) 8080 registers only.
ERZ80       ; Display Z80 registers.

EPRE        ; Pre-execution view of registers.
EPOST       ; Post-execution view of registers.

ETRACE      ; Trace all calls regardless of address.
-ETRACE     ; Do not trace calls to < 0100H.
E           ; Show current DebugZ status.
```

NOTE: Only the first three characters of each E command are examined, allowing you to abbreviate.

Example:

```
#E
All registers
Do not trace < 0100H
#
```

E-5. Fill memory

Summary: Fill memory with specific 8-bit data

Syntax: Fb,e,d

Description: Fill all memory within the range specified (inclusive) with the byte value.

Fb,e,d	Fill memory from location "b" through location "e" with data "d".

Example:

```
F155,.FOO+#50,'x'
```

E-6. Go

Summary: Execute code

Syntax: G[a][,[b][,c]]

Description: GO executes a program with up to two breakpoints:

G	Execute from the current PC (program counter)
G,b	Same as above with breakpoint at address "b".
G,b,c	Same as above with additional breakpoint at "c".
Ga	Execute starting at address "a".
Ga,b	Same as above with breakpoint at "b".
Ga,b,c	Same as above with additional breakpoint at "c".

-G...	Disable the normal display of pass points.for any command above
G,$	Break upon subroutine return (if no stacked values)

Example:

```
G100
G.START,3417,.ERR
-G,.EXIT+20
```

E-7. Hex Values

Summary: Display hex values or symbol table

Syntax: H[a][,b]

Description: The Hex command allows arithmetic operations in different bases as well as listing the symbol table:

Ha,b	Displays sum (a+b) and difference (a-b) in hexadecimal.
Ha	Displays result in up to four forms: hex #decimal 'char' .sym "char" is only displayed if it exists "sym" is the symbolic value, if any
H	Displays a list of all symbols; paginated.
-H	Same, except no page break pauses.

(NOTE: "a" and "b" are any legal expressions.)

Example:

```
H17+#34
H.FOO*.BAR^(#100+.GUANO)
```

E-8. Input Line

Summary: Enter command line and set memory locations

Syntax: Iline-of-text
Ifile1[file2]

Description: Initializes default buffer at 0080H, default FCB at 005CH, and default secondary FCB at 006CH to the "line-of-text" entered or to the names of the two files.

Example:

```
I FOO.COM FOO.SYM
ITEST.COM
I* TEST.SYM
```

NOTE: The primary filename is used by Read as the program to debug; the secondary filename is used by Read as the symbol table filename.

E-9. Full Screen Trace

Summary: Debug code in full-screen mode

Syntax: J

Description: Full trace mode is a powerful method to step through a program. There are four parts to the screen: memory display (4 lines), register display (2 lines), instruction display (16 lines), and command line area (2 lines). The memory area may be empty if the memory block (K) is not enabled. Other symbols include:

>	Instruction at which registers are displayed.
=	Current program counter
?:	Full screen command prompt

DebugZ retains up to 12 instructions worth of register values with a four instruction look-ahead; the memory window is updated automatically.

Non-execution control commands include:

< or ,	Move register pointer backwards
> or .	Move register pointer forwards
=	Set program counter
K	Set memory block view address (enable display)
+	Scroll memory view window forward by 32 bytes
-	Scroll memory view window backwards by 32 bytes
H	Expression evaluation (display at bottom)
?	Help
R	Redisplay the debug screen (if something affected it)
Q or X	Quit full-screen trace mode (return to '#' prompt)

Program execution instructions are:

space	Execute one instruction; no trace of CALLs
return	Execute one instruction; follow CALLs
1 to 5	Execute at 1 to 5 instructions/second; no trace of CALLs; any key halts
6 to 0	Execute at 1 (6) to 5 (0) instructions/second; follow CALLs; any key halts
Ga	Continue execution until address "a"; no display of intermediate registers/instructions.
Ga,b	Same, with an additional breakpoint at "b".

E-10. Memory View BlocK

Summary: Set block of memory to view in full-screen mode

Syntax: K[W]a
-K

Description: Sets initial bounds of memory block to view, if any, during full-screen trace (J) of a program.

Ka Sets the memory block to view from address "a" for up to 64 locations.
KWa Same, except view words rather than bytes.

-K Disables memory block view.

Only the starting address is entered. DebugZ will display as many values as will fit onto four lines of the screen.

Example:

```
K100
KW.FOO
```

E-11. List Code

Summary: List program code

Syntax: [-]L[b][,e]

Description: List assembly code including labels and symbols:

Lb	List disassembled code starting at "b" for 12 lines.
Lb,e	List disassembled code starting at "b" and ending at "e"; pauses for page breaks.
L	List disassembled code starting after last disassembled or traced address.
L,e	Same, except ending at address "e".
-L...	Same as above commands , except that pagination of the listed output is disabled.

Example:

```
L100
L,200
-L
```

E-12. Move Memory

Summary: Move memory

Syntax: Ms,e,d

Description: Move block of memory to another location:

Ms,e,d Move data from locations "s" through "e" to location "d" (through "d+e-s").

Either a head-to-tail or tail-to-head move is performed to copy source into destination avoiding memory overlap collision. Note that the source memory locations from "s" through "e" are not altered ***provided*** the destination block does not explicitly overwrite the source block during the copy operation.

Example:

```
M100,200,400
M3000,.FOOBIE,3100
```

E-13. Next Addresses

Summary: Display memory usage within DebugZ

Syntax: N

Description: DebugZ displays the memory usage when the command is entered. Address ranges utilized are displayed for:

- Prog Range occupied by program under test, if loaded.
- Free Free memory available for program or more symbols.
- Symbols Symbol storage used, if any.
- DebugZ Memory usage of DebugZ debugger program.

Example:

```
#N
Prog:    0100-04FF
Free:    0500-C4FF
Symbols: C500-C7D5
DebugZ:  C7D6-ED04
```

E-14. Pass Points

Summary: Set, reset, or display pass points

Syntax: P[p[,c]]
-P[p]

Description: A "pass point" is a program address to monitor during execution. Each pass point has a counter with a value from 1 to 255 indicating the number of times this address has been reached.

When the counter reaches 1, it stays there; any counters with a value of 1 are treated as "breakpoints". Program execution stops at any breakpoint. Up to eight passpoints are allowed.

Pp	Enter passpoint address "p"; pass count defaults to 1.
Pp,c	Same, except sets pass count to "c".

When reaching an active pass point during execution, the registers are displayed and the pass count , address, and symbol value (if any) are shown:

```
count PASS address .sym
8080 registers; instruction mnenomic
Z80 registers (if display of them enabled)
```

Display of pass points can be disabled during trace (T) , untrace (U) , and execution (G). Again, execution stops when the pass point is or becomes a value of one.

```
P
```

A list of the pass points is given in the following form:

```
pass-value pass-address   (symbol location, if any)
```

Pass points can be deleted by entering:

-Pa	where "a" is the pass address to remove.
-P	which deletes *all* pass points.

Example:

```
P.END
P.CONOUT,#37
P
```

E-15. Query Port

Summary: Read or write a port value

Syntax: QI[p]
QOp,v

Description: The Query command allows one to read and/or modify the value at a given port. A full 16-bit port value is accepted and used for both input and output. The format is:

QIp Display the value input from port "p".
QI Display the value input from port; the last port number "p" referenced is reused.

QOp,v Output 8-bit value "v" to port "p".

Example:

```
QIE0
QO45,'c'
```

E-16. Read File(s)

Summary: Read program and/or symbol file

Syntax: R[offset]

Description: The Read command loads a file into memory for debugging and/or reads a symbol table into DebugZ. The files are specified using the Input command. The format for the Read command is:

R Read executable file (if specified by Input) into memory starting at 0100H.
Read symbol file , if specified.

Ra Same, except Read executable file into memory at location "a" PLUS 0100H.
For example, "R200" reads the file in at location 0300H.
The offset is ignored when reading the symbol file, if any.

NOTE: See also Input.

Example:

```
R300
```

E-17. Set Memory

Summary: Set memory to string or value

Syntax: S[W]a

Description: Set memory alters the memory contents. Commands are:

Sa	Substitute bytes starting from location "a".
SWa	Substitute words starting from location "a".

In both cases, DebugZ prompts with the address and either 8-bit or 16-bit value at that location. One can either enter a new 8-bit (16-bit) value, press RETURN on a blank line to skip to the next location, or enter "." following by RETURN to quit.

Set can also be used to assign ASCII strings by beginning the line with single- or double-quotes (' or "). DebugZ strips the leading quote before setting memory to the string contents. Do NOT include a trailing quote; any trailing quote will be entered into memory.

Example:

```
S31FE
SW100
```

E-18. Trace Execution

Summary: Trace program execution displaying values

Syntax: T[W][n]

Description: Trace execution starting from current PC for the specified number of instructions. The flags and registers are displayed after each instruction. Subroutines (CALLs) are optionally traced.

T	Trace execution for one instruction including CALL statements.
Tn	Same, except trace for "n" instructions.
TW	Trace one instruction, do NOT trace CALLs.
TWn	Same, except trace for "n" instructions.

NOTE: Execution into addresses < 0100H may not be traced if the -ETRACE command has been specified (or is the default).

Example:

```
T
T5
TW#20
```

E-19. Untrace Execution

Summary: Trace *without* displaying intermediate values

Syntax: [-]U[W][n]

Description: Execution starting from current PC for the specified number of instructions. The flags and registers are *not* displayed after each instruction; only the final value of the flags and registers.

U	Untrace execution for one instruction including CALL statements.
Un	Same, except execute "n" instructions before a trace display
UW	Untrace one instruction, do NOT trace CALLs.
UWn	Same, except untrace for "n" instructions.
-U..	Same, except do NOT display pass points.

Note that CALLs to addresses < 0100H may not be traced if the -ETRACE command has been specified (or is the default).

E-20. Verify Memory

Summary: Compare memory locations

Syntax: [-]Vs,e,d

Description: Verify (compare) block of memory at one location to another:

Vs,e,d Verify data at locations "s" through "e" to location "d" (through "d+e-s"); paginates output.

-Vs,e,d Same except no pagination.

Any differences between the blocks are printed as:

```
src-addr src-value dst-addr dst-value
```

Example:

```
V100,200,3000
```

E-21. Examine/Alter Register Contents

Summary: Examine and/or alter Z80 registers

Syntax: X[r]
Xf

Description: The eXamine command can list the contents of all registers (and the current instruction) or alter any of them. Valid commands are:

X	Displays the current register contents, flags, and instruction.
Xr	Change register contents where "r" can be: "A", "B" for BC, "D" for DE, "H" for HL, "P" for PC, "S" for SP, "X" for IX, and "Y" for IY.
Xr'	Adding the "'" instructs DebugZ to alter the "backside" Z80 registers. Valid "r" include "A", "B", "D", and "H".

Changing the register value involves entering the new 16-bit value (8-bit for the "A" register). An empty line leaves the value unaltered. Altering the flags is done by entering the flag character (shown in **bold** below):

Xf	Change flags: **C**arry, **Z**ero, **M**inus, **E**ven parity
Xf'	Same, except alter the "backside" Z80 flags.

Entering a value of "1" at the prompt sets the flag, "0" resets it.

NOTE: Although any register or flag can be changed, display of the "backside" Z80 registers may optionally be suppressed (E) so you will not be able to see your changes on those registers.

Example:

```
XA
XX
XC
XE'
```

E-22. Search Memory

Summary: Search memory for specific values

Syntax: YBs,f
Y[q]

Description: Search allows searching for any sequence of bytes or characters, including wild cards.

YBs,f Sets bound of search between address "s" and address "f", inclusive.

Yq Search for sequence "q" starting from address "s".
Y Continue search for last sequence "q" starting from current address.

The sequence "q" is of the form:

```
value1,value2,....,value-n
```

where "value" is a byte, word, or string in quotes ("").

A question mark ("?") in a string is a wildcard that matches ANY byte.

Example:

```
YB100,2FFF
Y3E
Y'A',3F,"TEST_"
```

F. Summary

This is the quick reference section of the DebugZ documentation.

F-1. DebugZ Command Summary

Letter	Command	Syntax
A	Assemble	A[addr]
C	Call routine	Caddr
D	Display memory	[-]D[W][addr1][,addr2]
E	Extended commands	ERmachine [-]ETRACE
F	Fill memory	Faddr1,addr2,value
G	Go	G[addr][,break1[,break2]]
H	Hex arithmetic	H[val1[,val2]]
I	Input line	Istring
J	Full-screen trace	J
K	memory view block	K[W]addr -K
L	List code	[-]L[addr1[,addr2]]
M	Move memory	Mstart,end,dest
N	Next addresses	N
P	Pass points	P[addr[,count]] -P[addr]
Q	Query port	QI[port] QOport,value
R	Read file(s)	R
S	Set memory	S[W]addr
T	Trace execution	[-]T[W][instructions]
U	Untrace execution	[-]U[W][instructions]
V	Verify memory	[-]Vstart,end,dest
X	eXamine registers	X[reg] Xflag
Y	Search	YBstart,end Y[val1[,val2...]]

F-2. Full-screen Debug Command Summary

Letter	Command
< or ,	Move register pointer backwards
> or .	Move register pointer forwards
=	Set program counter
K	Set memory block view address (enable display)
+	Scroll memory view window forward by 32 bytes
-	Scroll memory view window backwards by 32 bytes
H	Expression evaluation (display at bottom)
?	Help
R	Redisplay the debug screen (if something affected it)
Q or X	Quit full-screen trace mode (return to '#' prompt)
space	Execute one instruction; no trace of CALLs
return	Execute one instruction; follow CALLs
1 to 5	Execute at 1 to 5 instructions/second; no trace of CALLs; any key halts
6 to 0	Execute at 1 (6) to 5 (0) instructions/second; follow CALLs; any key halts
Ga	No trace execution until address "a"
Ga,b	Same, with an additional breakpoint at "b".