SoftICE User's Guide

NuMega SoftICE

SoftICE 2.8
DOS

August 1997

Software License Agreement

Please Read This License Carefully.

You are purchasing a license to use NuMega Technologies, Inc. software. The software is owned by and remains the property of NuMega Technologies, Inc., is protected by international copyrights, and is transferred to the original purchaser and any subsequent owner of the software media for their use only on the license terms set forth below. Opening the package and/or using the software indicates your acceptance of these terms. If you do not agree to all of the terms and conditions, or if after use you are dissatisfied with the software, return the software, manuals and any partial or whole copies within thirty days of purchase to the party from who you received it for a refund, subject to our restocking fee.

Use Of The Software: NuMega Technologies, Inc. (“NuMega”), grants the original purchaser (“Licensee”) the limited rights to possess and use the NuMega Technologies, Inc. Software and User Manual (“Software”) for its intended purposes. Licensee agrees that the Software will be used solely for Licensee's internal purposes, and that at any one time, the Software will be installed on a single computer only. If the Software is installed on a networked system, or on a computer connected to a files server or other system that physically allows shared access to the Software, Licensee agrees to provide technical or procedural methods to prevent use of the Software by more than one user.

One machine-readable copy of the Software may be made for BACK UP PURPOSES ONLY, and the copy shall display all proprietary notices, and be labeled externally to show that the back-up copy is the property of NuMega, and that use is subject to this License. Documentation may not be copied in whole or part.

Use of the Software by any department, agency or other entity of the U.S. Federal Government is limited by the terms of the following “Rider for Governmental Entity Users.”

Licensee may transfer its rights under this License, PROVIDED that the party to whom such rights are transferred agrees to the terms and conditions of this Licensee, and written notice is provided to NuMega. Upon such transfer, Licensee must transfer or destroy all copies of the Software.

Except as expressly provided in this License, Licensee may not modify, reverse engineer, decompile, disassemble, distribute, sub-license, sell, rent, lease, give or in any way transfer, by any means or in any medium, including telecommunications, the Software. Licensee will use its best efforts and take all reasonable steps to protect the Software from unauthorized use, copying or dissemination, and will maintain all proprietary notices intact.

Limited Warranty And Indemnification: NuMega warrants the Software media to be free of defects in workmanship for a period of ninety days from purchase. During this period, NuMega will replace at no cost any such media returned to NuMega, postage prepaid. This service is NuMega's sole liability under this warranty.

NuMega agrees to indemnify and hold Licensee harmless from all loss, claim or damage to Licensee arising out of any claim, legal action, or suit alleging that the Software infringes a United States patent, copyright, or trade secret. NuMega will defend, at its own expense, any such claim or action against Licensee, and will pay all reasonable costs, expenses, and damages incurred by Licensee in connection therewith, including reasonable attorneys' fees and expenses incurred by Licensee, on condition that NuMega shall be immediately notified in writing by Licensee of any notice of such claim or action or pending claim or action known to Licensee prior to the time when the failure to deliver such notice has injured NuMega; and that NuMega shall have control of the defense against any such claim or action and all negotiations toward the settlement or compromise thereof. In such event, Licensee shall have the right to participate in any such action at NuMega's cost. If the Software becomes the subject of any claim, suit, or proceeding for infringement of any United States patent, copyright or any other right of a third party, or in the event of any adjudication that the Software infringes upon any United States patent, copyright or any other third party, or if the use or license of such Software is enjoined, in addition to other liability which may arise under this provision, NuMega shall at its sole expense and discretion either (a) obtain a license or any rights necessary to make the warranty contained herein true and correct, or (b) refund to Licensee any amounts paid to NuMega for the Software. Specifically excluded from the above covenant are any claims or actions based upon, or portions of claims or actions based upon, those portions of the Software modified or developed by Licensee or third parties.

Disclaimer: LICENSE FEES FOR THE SOFTWARE DO NOT INCLUDE ANY CONSIDERATION FOR ASSUMPTION OF RISK BY NUMEGA, AND NUMEGA DISCLAIMS ANY AND ALL LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR OPERATION OR INABILITY TO USE THE SOFTWARE, EVEN IF ANY OF THESE PARTIES HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. FURTHERMORE, LICENSEE INDEMNIFIES AND AGREES TO HOLD NUMEGA HARMLESS FROM SUCH CLAIMS. THE ENTIRE RISK AS TO THE RESULTS AND PERFORMANCE OF THE SOFTWARE IS ASSUMED BY THE LICENSEE. THE WARRANTIES EXPRESSED IN THIS LICENSE ARE THE ONLY WARRANTIES MADE BY NUMEGA AND ARE IN LIEU OF ALL OTHER WARRANTIES EXPRESSED OR IMPLIED, INCLUDING BUT NOT LIMITED TO IMPLIED WARRANTIES OF MERCHANTABILITY AND OF FITNESS FOR A PARTICULAR PURPOSE.

THIS WARRANTY GIVES YOU SPECIFIED LEGAL RIGHTS, AND YOU MAY ALSO HAVE OTHER RIGHTS WHICH VARY FROM JURISDICTION TO

JURISDICTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF WARRANTIES, SO THE ABOVE LIMITATIONS OR EXCLUSIONS MAY NOT APPLY TO YOU.

Term: This License is effective as of the time Licensee receives the Software, and shall continue in effect until Licensee ceases all use of the Software and returns or destroys all copies thereof, or until automatically terminated upon the failure of Licensee to comply with any of the terms of this License.

General: This License is the complete and exclusive statement of the parties' agreement. Should any provision of this License be held to be invalid by any court of competent jurisdiction, that provision will be enforced to the maximum extent permissible, and the remainder of the License shall nonetheless remain in full force and effect. This License shall be controlled by the laws of the State of New Hampshire, and the United States of America.

Rider For U.S. Government Entity Users

This is a Rider to the above Software License Agreement, (“License”), and shall take precedence over the License where a conflict occurs.

1.The Software was: developed at private expense; no portion was developed with government funds; is a trade secret of NuMega and its licensor for all purposes of the Freedom of Information Act; is “commercial computer software” subject to limited utilization as provided in any contract between the vendor and the government entity; and in all respects is proprietary data belonging solely to NuMega and its licensor.

2.For units of the DOD, the Software is sold only with “Restricted Rights” as that term is defined in the DOD Supplement to DFAR 252.227-7013 (b)(3)(ii), and use, duplication or disclosure is subject to restrictions set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at 252.227-7013. Manufacturer: NuMega Technologies, Inc. P.O. Box 7780, Nashua, New Hampshire 03060-7780 USA.

3.If the Software was acquired under a GSA Schedule, the Government has agreed to refrain from changing or removing any insignia or lettering from the Software or Documentation or from producing copies of manuals or disks (except for back up purposes) and; (1) Title to and ownership of the Software and Documentation and any reproductions thereof shall remain with NuMega and its licensor; (2) use of the Software shall be limited to the facility for which it is acquired; and (3) if the use of the Software is discontinued at the original installation and the Government wishes to use it at another location, it may do so by giving prior written notice to NuMega, specifying the new location site and class of computer.

4.Government personnel using the Software, other than under a DOD contract or GSA Schedule, are hereby on notice that use of the Software is subject to restrictions that are the same or similar to those specified above.

Contents

1 Product Description

SoftICE is a software debugging tool that provides hardware-level debugging capabilities to PCDOS and MSDOS debuggers.

SoftICE uses 80386 protected mode to run DOS in a virtual machine. This gives SoftICE complete control of the DOS environment. SoftICE uses 80386 protected mode features, such as paging, I/O privilege level, and break point registers, to add hardware-level break points your existing DOS debugger.

SoftICE was designed with three goals in mind:

1. To utilize the 80386 virtual machine capability to debugging features that are impossible or prohibitively slow with software-only debuggers (e.g., real time hardware-level break points, memory protection, breaking out of hung programs, etc.).

2. To work with existing debuggers. We wanted to provide a tool that worked with existing tools. We designed SoftICE in such a way that you don't have to learn a new debugger to get powerful hardware debugging capabilities.

3. To be a user-friendly program with a window that pops up instantly and does not get in the way. All of the SoftICE commands were designed to fit in a small window so that information on the screen behind SoftICE could still be viewed. Dynamic on-line help assists users who only use SoftICE occasionally.

The SoftICE program features:

- real time break points on memory reads/writes, port reads/writes, memory ranges, and interrupts
- back trace history ranges
- symbolic and source level debugging
- an environment that works with existing debuggers
- full EMM 4.0 support

- backfilling to raise base memory past 640K for monochrome systems
- a window that can pop up at any time
- the ability to break out by keystroke even if interrupts are disabled
- debugger code that is isolated by 80386 protected mode. This prevents an errant program from modifying or destroying SoftICE; even if DOS clobbered, SoftICE will still work
- the ability to configure SoftICE to use no memory in the lower 640K if the system has more than 640K
- user-friendly dynamic help
- the ability to be used as a stand-alone debugger. This ability is useful if you are debugging loadable device drivers, interrupt handlers, or boot sequences where traditional debuggers can't go, if your debugger suffers from re-entrancy problems
- a soft boot capability that allows debugging with non-DOS operating systems or self-booting programs
- a simple installation, with no DIP switches to set no I/O ports taken up, and no memory address space conflicts
- integration with BoundsChecker
- ability reads symbolic and source information directly from the .EXE header from Microsoft & Borland languages
- overlay support for Microsoft's LINK and Pocket Soft's .RTLink/Plus
- two symbol tables loaded at the same time
- 386 32-bit instruction dis-assembly and 32-bit register dump
- Microsoft C version 6 compatible
- numeric processor dis-assembly
- allows device drivers and T&SR programs to load high
- VCPI support
- remote debugging
- 80486 support

Note: SoftICE will work with real address mode programs only. It will not work with programs that use 80286 or 80386 protected mode instructions.

System Requirements

SoftICE works with the IBM Series II Model 70 and 80, Compaq 80386 and 80386SX computers, AT compatible and 80386 co-processor cards. SoftICE will only work with 80386 XT co-processors if they are AT compatible.

SoftICE works best with extended memory, but works fine with conventional memory systems.

SoftICE does not use DOS or ROM BIOS for its video output and keystroke input. Therefore the video must be compatible with one of the following: MDA, Hercules, CGA, EGA, or VGA. SoftICE also has support for a two- monitor configuration, which can be very helpful when debugging video- intensive programs.

2 Getting Started

The Diskettes

When you run SoftICE, the name of the person that your copy of SoftICE is licensed to is displayed on the screen as a deterrent to software pirates. The SoftICE diskette is not physically copy-protected for your convenience. For our convenience, we appreciate your high regard for our licensing agreement. It is important to make a duplicate copy to be used only for backup in case the original diskette is damaged.

A directory of a SoftICE diskette will show the following files:

```
S-ICE.EXE
S-ICE.DAT
LDR.EXE
MSYM.EXE
EMMSETUP.EXE
UPTIME.EXE
README.SI
SAMPLE.EXE
SAMPLE.ASM
SAMPLE.SYM
LH.EXE
LD.SYS
ADDHI.EXE
CE.EXE
IOSIM.ASM
```

S-ICE.EXE is the SoftICE program.

S-ICE.DAT is the SoftICE initialization file.

LDR.EXE is the SoftICE program and symbol file loader.

MSYM.EXE is the SoftICE symbol file creation program.

EMMSETUP.EXE is a program that allows you to customize the way your system will use expanded memory.

UPTIME.EXE sets the time to that of the real time clock.

README.SI is a text file containing information about SoftICE that did not make it into this manual.

SAMPLE.EXE is a short demonstration program that is used with the tutorial.

SAMPLE.ASM is the assembly language source file for the demonstration program.

SAMPLE.SYM is the symbol file for the demonstration program.

LH.EXE is a utility that loads high T&SRs.

LD.SYS is a utility that loads high DOS loadable device drivers.

ADDHI.EXE is a utility that adds high memory to DOS memory chain.

CE.EXE is the CONFIG.SYS editor.

IOSIM.ASM is an example of a user qualified break point. It will take a BPIO break point and log all the values that were written to or read from that port.

Loading SoftICE

Before running SoftICE, copy all of the files on the distribution diskette to your hard disk.

These files should be placed in a directory that is accessible through your alternate path list.

S-ICE.EXE can be loaded as a device driver in CONFIG.SYS or can be run as a program from the command line. To use many of SoftICE's features, S- ICE.EXE must be loaded as a device driver in CONFIG.SYS.

Note: If you do not have extended memory, SoftICE can NOT loaded as a device driver. Instead, it must be run from the DOS prompt.

Loading Without Extended Memory

When no extended memory is present, SoftICE loads it at the highest memory location possible. The memory used by SoftICE is then 'mapped out', making it invisible to DOS programs. Since the total memory visible to DOS its programs is less after SoftICE loads, it is recommended that you load SoftICE before any TSR's or control programs. If you do not have extended memory, simply enter:

```
S-ICE
```

Loading With Extended Memory

Loading SoftICE with extended memory can be done in one of two ways:

1. Install S-ICE,EXE as a driver in CONFIG,SYS, This method is necessary if you will be using one the following capabilities:

   - Sharing memory with programs that use extended memory by using ROM BIOS calls (VDISK,SYS, RAMDRIVE.SYS, HIMEM.SYS, cache programs, etc.).
   - Using SoftICE's EMM 4,0 capability,
   - Using SoftICE for symbolic or source level debugging.
   - Using back trace ranges.
   - Using SoftICE with other Nu-Mega products such as MagicCV

   When loaded as a driver, SoftICE allocates a portion of extended memory for itself and its associated components so there can be no memory conflicts. S-ICE.EXE must be loaded in CONFIG.SYS before any other driver that allocates extended memory is loaded (e.g., VDISK.SYS, RAMDRIVE.SYS). Generally SoftICE works best if it is the first loadable device driver installed in CONFIG.SYS.

   For users that are new to SoftICE it is advisable to load SoftICE as the first driver in CONFIG.SYS with the following statement:

   device = drive: \path\S-ICE.EXE /SYM 50

   Drive and path specify the directory where S-ICE.EXE is located. This statement will load SoftICE at system initialization and will be adequate for the tutorial. However, SoftICE will not be installed for some of its more powerful features such as EMM 4.0. You can reconfigure SoftICE with those features enabled after you have experimented a bit. If you already have experience with SoftICE or would like to set up SoftICE with those features immediately, please read chapter 6 (SoftICE Initialization Options).

Caution: When installing any new device driver for the first time on your system, it is advisable to have a boot diskette available. This precautionary measure is for the unlikely event that the default setup of the device driver is not compatible with your system.

   If you are not sure how to edit your CONFIG.SYS file, refer to your system user's guide or your text editor user's guide for instructions. After you have modified your CONFIG.SYS file, you must reboot your system to have the changes take effect.

2. Run SoftICE from the DOS Prompt by typing S-ICE. Before actually loading, SoftICE will display a loading message and prompt. To prevent this prompt, place the word EXTENDED in the S-ICE.DAT file. See *The SoftICE Initialization File S-ICE.DAT* on page 144 for more information on the S-ICE.DAT file. Using this method, S-ICE.EXE

is automatically loaded into the top of extended memory, whether or not anything else is already there. If you know you will not have any other programs using extended memory, this method is acceptable. When loaded with this method, SoftICE occupies ZERO bytes of conventional memory. The command you use is: S-ICE

Note: You can NOT enable all of SoftICE's features when Loading from the command line. If you will be using SoftICE as a stand-alone debugger, it is recommended to Load SoftICE from CONFIG.SYS. If you want to load SoftICE as a device driver, but don't want SoftICE to be resident all of the time, you should use the /UN loading switch. Refer to *SoftICE Loading Switches* on page 143 for more information.

Configuring SoftICE for a Customized Installation

You can customize SoftICE with SoftICE loading switches in CONFIG.SYS and with the SoftICE initialization file S-ICE.DAT. The CONFIG.SYS loading switches allow you to customize how the extended memory will be reserved by SoftICE. The initialization file S-ICE.DAT allows you to specify configuration options, assign commands to function keys, and define an auto-start string. An auto-start string is used to execute a series of commands that you use every time you install SoftICE. for more information about customizing SoftICE, refer to *Chapter 5: SoftICE Initialization Options* on page 141.

Unloading SoftICE

Occasionally you may need to unload SoftICE. A typical reason for unloading SoftICE is to run a program that uses 80286 or 80386 protected mode instructions. To unload SoftICE, enter:

```
S-ICE /U
```

This command places the machine back in real address mode. If SoftICE was initially loaded from CONFIG.SYS When the memory is still reserved for SoftICE and can not be used by other software. If SoftICE was initially loaded from the command line, unloading frees up the memory consumed by S- ICE.EXE.

Caution: If you have any backfilled memory in your system, or if expanded memory is currently being used, unloading SoftICE could crash your system.

Reloading SoftICE

SoftICE can be re-loaded at any time even if it had initially been loaded in CONFIG.SYS. If SoftICE had initially been loaded in CONFIG.SYS then the original configuration options (EMM 4.0, symbols and source...) are still in effect. To reload SoftICE, enter:

```
S-ICE
```

3 Debugging in 30 Minutes

Introduction

All interaction with SoftICE takes place through a window that can be popped up at any time. All SoftICE commands fit in a small window, but the window can be enlarged to full screen. You will typically use the small window when using SoftICE as an assistant to another debugger, and the large window when using SoftICE in stand-alone mode.

The window initially comes up in full screen mode if you are using the SoftICE configuration file (S-ICE DAT) that was included on the distribution diskette.

Popping Up the Window

You can bring up the window at any time after installing SoftICE. You initially bring up SoftICE by pressing the CTRL and D keys. However, this sequence can be changed by using the ALTKEY command (see *ALTKEY* on page 114).

Returning From the Window

Return to the original display by using the X command or the key sequence that you used to invoke SoftICE. Any break points that you set while working in SoftICE will be armed at this point.

Changing the Window Size

You can modify both the width and the height of the SoftICE window. Changing the window size is particularly useful in stand-alone mode when you are displaying code memory.

The window height can vary from 8 to 25 lines tall. To change the window height, use the following key sequences: ALT ? -- makes the window taller ALT ? -- makes the window shorter

To change the window width, use the WIN command (see *WIN* on page 129). Entering WIN with no parameters toggles between the following two modes:

WIDE mode -- full screen width

NARROW mode -- 46 characters wide

Some commands (i.e., D, E, R, U) take advantage of the extra width by displaying more information when the window is in wide mode.

Moving the Window

The SoftICE window is movable and can be positioned anywhere on the screen. This is particularly useful when the window is in narrow mode. Move the window anytime you need to view information on the screen behind the window. The following key sequences move the window:

CTRL Up Arrow -- moves the window one row up

CTRL Down Arrow -- moves the window one row down

CTRL Right Arrow-- moves the window one column right

CTRL Left Arrow-- moves the window one column left

Line Editing Keystrokes

SoftICE's easy-to-use line editor allows you to recall and edit previous commands. The line editor functions are similar to those of the popular CED line editor. The following key sequences help you edit commands in the command window:

RIGHT ARROW-- moves the cursor to the right

LEFT ARROW-- moves the cursor to the left

INS-- toggles insert mode

DEL-- deletes the current character

HOME-- moves the cursor to start of the line

END-- moves the cursor to the end of the line

UP ARROW-- displays the previous command

DOWN ARROW-- displays the next command

SHIFT+UP ARROW-- scroll one line up in display

SHIFT+DOWN ARROW-- scroll one line down in display

PAGE UP -- scroll one page up in display

PAGE DN -- scroll one page down in display

BKSP -- deletes the previous character

ESC-- cancels the current command

There are special key assignments when the cursor is in the data window or the code window. These are described in the sections for the E and EC command respectively. One special assignment of note is the SHIFT ? and Shift ? keys while the cursor is in the code window. These keys are re- assigned so they have the functions that ? and ? normally have. This way you can recall previous commands while the cursor is in the code window.

Interactive Status Line

A status line at the bottom of the window provides interactive help with command syntax.

Command Syntax

SoftICE is a command-driven debugging tool. To interact with SoftICE, you enter commands, which can optionally be modified by parameters.

All commands are text strings that are one to six characters in length and are case insensitive. AlI parameters are either ASCII strings or expressions.

Expressions are typically numbers, but can also be combinations of numbers and operators (e.g., + - /*). All numbers are displayed in hexadecimal format. Byte parameters are 2 digits long, word parameters are 4, and double word parameters are 2 word parameters separated by a colon (:). Here are some examples of parameters:

- 12 -- byte parameter
- 10FF -- word parameter
- E000:0100 -- double word parameter

Registers can be used in place of bytes or words in an expression. For example, the command 'U CS:IP-10' will start unassembling instructions ten bytes before the current instruction pointer address. The following register name may be used in an expression:

AL, AH, AX, BL, BH, BX, CL, CH, CX, DL, DH,

DX, DI, SI, BP, SP, IP, CS, DS, ES, SS, or FL

Specifying Memory Addresses

Many SoftICE commands require memory addresses as parameters. A memory address is a value that is made of two 16-bit words, separated by a colon. The first word is the segment address, and the second word is the segment offset.

Public symbols can be used in place of an address in any SoftICE command. The public symbols must have been loaded with the SoftICE program loader (LDR.EXE). See *Chapter 6: Symbolic and Source Level Debugging* on page 149 for a complete description of using public symbols.

The SoftICE expression evaluator recognizes several special characters in conjunction with addresses. These special characters are:

- $ -- Current CS:IP.
- @address -- Double Word Indirection
- .number -- Source Line Number

The $ character can be used in place of CS:IP when typing the address of the current instruction pointer.

The @ character allows you to refer to the double word pointed to by the address. You can have multiple levels of @'s.

If the . character precedes an address, the address will be interpreted as a source line number in the current file, rather than an actual address. This is only valid when source files are loaded. The address is interpreted as a decimal number in this case.

Examples:

U.1234

- This command starts unassembling instructions at source line 1234 decimal.

U $-10

- This command unassembles instructions starting 10 bytes prior to the current instruction pointer.

G @SS:SP

- Assume you are at the first instruction of an interrupt routine. Entering this command will set a temporary break point at the return address on the stack and skip the interrupt routine.

Function Keys

Function keys can be assigned to any command string that can be typed into SoftICE. Function keys can be assigned from the command line or pre- initialized through the SoftICE definition file S-ICE.DAT.

The default S-ICE.DAT that comes on the SoftICE distribution diskette has definitions for all 12 function keys. You can change any of these definitions at any time. They are intended as examples, but they are designed to make easy for users of Microsoft's CodeView, Thee default assignments are:

F1 -- Displays general help (H;)

F2 -- Toggles the register window (^WR;)

F3 -- Changes current source mode (^SRC;)

F4 -- Restores screen (^RS;)

F5 -- Returns to your program (^X;)

F6 -- Toggles cursor between command window code window (^EC;)

F7 -- Goes to current cursor line (^HERE;)

F8 -- Single steps (^T;)

F9 -- Sets break point at current cursor line (^BPX;)

F10 -- Program steps (^P;)

F11 -- Go to return address (large model) (^G@SS:SP;)

F12 -- Displays SoftICE version number (^VER;)

A caret (^) preceding a command makes it invisible, a semi-colon (;) following a command represents a carriage return. You can display the current function key assignments by entering the command:

FKEY

To use a function key simply press the function key instead of entering the command. To program function keys see *FKEY* on page 115 for a description of the FKEY command, or *Chapter 5: SoftICE Initialization Options* on page 141 for a description of pre-initializing function keys in S-ICE.DAT.

Help

The help command displays a short description, a syntax expression, and an example of each command. To display help information, enter:

? or H -- displays short descriptions of all commands and operators

? command or H command -- displays more detailed information on the specified command, syntax, and an example

? expression or H expression -- displays the value of the expression in hexadecimal, decimal and ASCII

Tutorial

The following tutorial demonstrates a few of the features SoftICE and gives you the opportunity to try using SoftICE. SoftICE can be used in conjunction with another debugger or as a stand-alone debugger. The tutorial demonstrates using SoftICE as an assistant to the DOS debugger, DEBUG, and then shows how SoftICE can be used as a stand-alone debugger with source and symbols loaded. DEBUG can be found on the PCDOS or MSDOS system diskette. If you do not have DEBUG, you can use another debugger in its place, or SoftICE can be used as a stand-alone debugger.

Users who need to use SoftICE for a reverse engineering project, or for debugging DOS loadable device drivers or Terminate and Stay Resident programs should go through this tutorial too. Even though examples of these types of programs are not demonstrated directly, you will get an overview of debugging with SoftICE. It is recommended that you experiment with SoftICE and your particular environment before beginning a real project.

A short assembly language program with a subtle flaw is used to demonstrate hardware-style break points. The sample program has been kept intentionally short and to-the-point for those not very familiar with assembly language. The tutorial is designed to give you a peek at SoftICE features. Feel free to experiment on your own after going through the tutorial.

Since SoftICE is very flexible, it allows you to load in the way that is best for your system. Go through the installation procedures in section 2.2 before continuing with the tutorial.

If you do not have extended memory on your system, you must load SoftICE from the command line. When loading SoftICE from the command line you can not load symbols or source files. In this case you must improvise in the last section of the tutorial where SoftICE is used as a stand-alone debugger.

SoftICE can be loaded from the DOS prompt or loaded as a device driver in CONFIG.SYS. For the purpose of this tutorial you should install SoftICE in CONFIG.SYS with at least 50K of extended memory reserved for symbols and source files. SoftICE should be the first device driver installed in CONFIG.SYS. The device installation line should look like:

```
DEVICE = drive: path\S -ICE.EXE /SYM 50
```

The /SYM 50 parameter instructs SoftICE to reserve 50 kilobytes of extended memory for symbols and source file This is not enough to solve most real world problems, but will work for our sample program.

You must re-boot your system after placing this line in CONFIG.SYS.

When you re-boot your system SoftICE displays a copyright notice, a registration number, the name of the person who owns this copy of Soft- ICE, and the amount a extended memory reserved for each SoftICE component. On a system with 384K of extended memory the initial screen looks like:

SoftICE

Your Name Your Company Name Registration # SInnnnnn

Copr. (C) Nu-Mega Technologies 1987-1989

All Rights Reserved

SoftICE Version 2.00

SoftICE is loaded from 00132000H up to 00160000H.

50K of symbol space reserved.

10K of back trace space reserved.

200 K of extended memory available.

The “SoftICE is loaded ...” message tells you the exact area of memory that SoftICE and its components are occupying. If you are on a Compaq or Compaq clone and have included the word COMPAQ in your S-ICE.DAT file you would also see a message saying “Using high memory from XXXXXXXX to 00FE0000H”.

The next line tells you how much symbol space has been reserved. This space is used for both symbols and source files.

The next line tells you how much memory has been reserved for back trace history. This amount defaults to 10K. This memory area is used by the SNAP command and the BPR command with the T or TW options.

The last line tells you how much memory is left for regular extended memory. This memory can be used by other programs, such as HIMEM, SMARTDRIVE, VDISK, etc.

Change directories to the hard drive directory where you loaded all the files from your distribution diskette. Remember, this directory must be accessible from your alternate path list.

Before we get into heavy debugging, let's bring the SoftICE window up and give it a test drive.

Clear the screen by entering:

◊ CLS

Bring up the SoftICE window by pressing:

◊ CTRL D

The SoftICE window is now on the screen. If you have file S-ICE.DAT accessible from your path then the SoftICE window will occupy the entire screen. It will be divided into four sections. From top to bottom, these sections are the register window, the data window, the code window, and the command window. If S-ICE.DAT was not found then you will have a small window in the center of the screen. This also means that other components needed for the tutorial have not been loaded.

If the small window is visible you should:

1. Exit from SoftICE by entering X.
2. Unload SoftICE by entering S-ICE /U.
3. Copy the file S-ICE.DAT from the distribution diskette to a directory accessible from your current path.
4. Restart the demo.

We will now switch to the small window. The small window is very convenient for using SoftICE as an assistant to another debugger.

Enter:

◊ WIN

This will make a small command window in the center of the screen. Several SoftICE commands are visible on this screen. These are remnants of the initialization string in S-ICE.DAT that originally set up SoftICE in the full screen mode.

You will notice a prompt symbol (:) and a status line at the bottom of the window.

The SoftICE window can be moved around on the screen, and the window size can be adjusted.

Move the window around the screen by pressing:

- CTRL ? -- moves the window up one row
- CTRL ? -- moves the window down one row
- CTRL -- moves the window one column left
- CTRL ? -- moves the window one column right

Change the window size so that it fills the whole screen by entering:

- WIN

You will notice that the original screen is back.

Change back to the small window by entering WIN again.

Make the window taller or shorter by pressing:

- ALT ? -- makes the window taller
- ALT ? -- makes the window shorter

Now try what comes naturally when you're in front of a new program and you don't have the foggiest notion of what to do next -- ask for help.

Get a help display by entering:

?

Notice how the display stops and waits for a keystroke before scrolling any information off the screen. Look at the status line at the bottom of the window. The status line displays the instructions: "Any Key To Continue, ESC to Cancel ". Now press any key to continue displaying more the help information. Continue pressing the key until the prompt (:) reappears.

Scroll back through the help information by pressing

- SHIFT + UP ARROW

Previously displayed information in the command window can be scrolled with the shift up, shift down, page up and page down keys. Try a variety of these keys to scroll through the help information.

The SoftICE help facility gives you an overview of each command. If you enter a question mark (?) followed by a command name, you see a display showing the command syntax, a short description of the command, and an example.

Try experimenting with help by entering commands in this format:

◊ ? command

For example,

◊ ? ALTKEY

Pay attention to the status line prompts on the bottom line of the screen if you get confused.

The help command also allows you to evaluate hexadecimal expressions.

For example, enter:

◊ ? 10*2+42

The resulting display shows you the value of the expression, first in hexadecimal, then decimal, then in ASCII representation:

◊ 0062 00098 “b”

We brought up the window with the CTRL D key sequence. That's all right for some, but you may prefer to use another key sequence.

We are now going to enter a command to change the key sequence required to bring up the window. We'll do this one step at a time, so you can get used to the status line at the bottom of the window.

Type the letter ‘A’. The status line displays a list of all the commands starting with the letter ‘A’. Finish typing the word ‘ALTKEY’. The status line now displays a short description of the /ALTKEY command Press the space bar. The status line now shows the required syntax for the /ALTKEY command. Type the letters ‘ALT D’ then press ENTER to enter the entire command:

◊ ALTKEY ALTD

You just changed the window pop up key sequence to ALT D. From now on, you must press the ALT D key sequence to pop up the window. This is assumed throughout the remainder of the tutorial.

Now let's test the previous command.

To exit from the window, press:

◊ ALT D

The SoftICE window just disappeared.

To return to the SoftICE window, release the ALT key, then press:

◊ ALT D

The window returned.

To see some previous commands, press:

◊ the UP ARROW key a few times.

Notice that SoftICE remembers commands that have been entered. Try editing one just for fun. Some of the editing keys are:

INS -- Toggles insert mode on or off

DEL -- Deletes one character

HOME -- Moves the cursor to start of line

END -- Moves the cursor to end of line

RIGHT ARROW-- Moves the cursor one column to the right

LEFT ARROW-- Moves the cursor one column to the left

When insert mode is on, notice that the cursor is in a block shape.

Now that you are somewhat familiar with the environment let's try some more commands.

Erase the command you were editing by pressing the HOME key, then pressing the DEL key until the command is gone.

Enter:

◊ WR

The WR command makes the register window visible. The register window displays the contents of the 8086 registers. Notice that the register values reflect the location where the code was executing when you invoked SoftICE.

The WR command is assigned to the function key F2 in the SoftICE initialization file S-ICE.DAT.

Press the F2 key several times and you will see the register window toggle on and off. Leave the register window visible.

Extend the vertical size of the SoftICE window by holding down the ALT and the until the window is the entire length of the screen. Notice the values of the CS and IP registers in the register window, then enter:

◊ MAP

The MAP command displays a system memory map. The area of the current instruction pointer (CS:IP) is highlighted. If you have a complex memory map you may have to press a key a few times until the until the prompt reappears.

Now try the following sequence a few times, noticing the (CS:IP) registers in the register window.

- ALT D
- Release ALT and D
- ALT D

Each time you bring the SoftICE window back up you will notice that the CS and IP registers have changed. When CS and IP change you can enter the MAP command again to see if the instruction pointer now points to a different area.

This little exercise demonstrates that SoftICE is a system level debugger that pops up wherever the instruction pointer happens to be when you press the SoftICE hot key sequence. The instruction pointer is continuously changing because there is a lot of activity happening behind the scenes even when you are at the DOS prompt, such as timer interrupts, DOS device driver polling, DOS busy waiting other interrupts, etc.

Press the F12 function key.

The F12 function key defaults to be assigned to the SoftICE VER command. It displays the SoftICE copyright message and the version number. We will now assign the F12 function key to the SoftICE RS command.

Enter:

- RS

This will temporarily show the program screen without the SoftICE window.

Press the space bar to get back to get back the SoftICE window.

Enter:

- FKEY F12 RS;

This assigns the RS command to the F12 key. The semi-colon represents the ENTER key.

Press the F12 key.

Repeat this a few times to toggle between the SoftICE window and the program screen. Now make sure the SoftICE window is displayed, by pressing the F12 key if necessary. You will notice RS displayed several times in the window. There is one occurrence for each time you pressed the F12 key to show the program screen.

Clear the SoftICE window by entering:

◊ CLS

Enter:

◊ FKEY F12 ^RS;

The ^ symbol is a shifted 6. This assigns the RS command to the F12 key, but makes it an invisible command.

Press the F12 key several times. Notice that the RS command no longer displays in the SoftICE window.

You can also assign a sequence of SoftICE commands to a function key. Remember to place a carriage return between each command. Now let's prepare to use SoftICE as an assistant to the MSDOS DEBUG utility.

Get rid of the register window by pressing the F2. then shrink the window size down to about 6 lines by Using ALT .

Enter:

◊ ACTION INT3

This command tells SoftICE to generate interrupt 3's when break point conditions are met. That's how SoftICE will communicate with DEBUG. The default setting is HERE. ACTION HERE will cause control to return directly to SoftICE. Use ACTION HERE when using SoftICE as a stand-alone debugger.

For those of you not using DEBUG with this tutorial you might have to improvise now. CODEVIEW works with ACTION set to NMI. Most other debuggers will work with ACTION set to INT3. If your debugger doesn't, and you need help improvising, refer to the complete description ACTION (see *ACTION* on page 81).

To make the SoftICE window disappear again, enter:

◊ X

This is an alternative method to exit from SoftICE. This especially useful in function key definitions.

Now that you are familiar with some of the basics of using SoftICE, let's learn some details by debugging the sample program (SAMPLE.ASM).

SAMPLE.ASM is a simple program written in assembly language by a programmer named Jed. The program reads a keystroke from DOS and displays a message telling whether the keystroke was a space.

To run the program SAMPLE, enter:

- SAMPLE

Now press the space bar. Press several keys. Jed's program obviously has a problem! Jed has spent hours studying this source code and is certain there are no flaws in his logic. However, Jed borrowed some 'helper' routines from his friend Jake (get_key, is_space?). Jed is somewhat suspect these routines but he cannot find the bug.

The source code for Jed's program looks like this:

```
Page 55,80
Title Sample program for SoftICE tutorial
DATA Segment Public 'Data'
pad db 12H dup(0)
char db 0
answer db 0
Space_msg db 'The Character is a SPACE',0DH,0AH,'$'
no_space_msg db 'The Character is NOT a' db 'SPACE',0DH,0AH,'$'
DATA Ends

STACK Segment Stack 'Stack' Dw 128 Dup (?) ;Program stack
STACK Ends

CODE Segment Public 'Code'

Assume CS:CODE,DS:DATA,ES:Nothing,SS:STACK

start:

; Set up segments

    mov ax,DATA
    mov es,ax
    mov ds,ax

; Main Program Loop

main,loop:
    call get_key
    call is_space?
    cmp answer,0
    je no,space
; It's a space, so display the space message
    mov ah,9
    mov dx,offset space_msg
    int 21H
```

```
    jmp main_loop

; It's NOT a space, so display the no space message

no_space:
    mov ah,9
    mov dx,offset no_space_msg
    int 21H
    jmp main_loop



;-------------------------------------------------------;;
```

JAKE'S ROUTINES

```
;-------------------------------------------------------;
; Get Key Routine (one of Jake's routines)
get_key proc
    mov ah,8
    int 21H
    mov char,al
    ret
get_key endp

; Check if character is a space (one of Jake's routines)

is_space?proc
    cmp char,20H
    jne not_space
    mov answer, 1
    ret
not_space: mov cs:answer,0
    ret
is_space?endp

CODE Ends
Endstart
```

Jed has been using DEBUG but has not been able to pinpoint the problem. As a recommendation from his nephew Jethro, Jed has purchased SoftICE. He was somewhat reluctant to use it because he had tried a hardware-assisted debugger but could never get it working quite right. He was willing to try SoftICE because he could continue to use DEBUG -- the only debugger he really understood.

Press CTRL C to break out of the program.

Enter the following commands:

- DEBUG drive:\pathname\SAMPLE. EXE
- U
- R

In the hours Jed has spent trying to find this elusive bug, he has had the suspicion that something is overwriting his code in some subtle way. With SoftICE, Jed decides to set a range break point across his code segment.

Press:

- ALT D

The SoftICE window is back. Move the window (by using CTRL and the arrow keys) until DEBUG's register display is visible. It's time to set our first break point.

Enter:

- BPR code-seg:0 code-seg:25 W

Code-seg is the value in the CS register as displayed by the DEBUG R command.

The BPR command sets a memory-range break point. The length of Jed's code segment is 25H bytes, so the memory range specified goes from the beginning of his code segment to the end. The W tells SoftICE to break on a write. We want to catch any unexpected writes to Jed's code.

Enter:

- BL

The BL command displays all break points. The display from BL looks similar to the following display:

0) BPR code-seg:0000 code-seg:0025 W C = 01

The 0 is the identifier for this break point. The range and W are displayed as they were entered, and the count (since none was specified) defaults to one.

Now comes the moment of truth.

Press

- ALT D.

The window disappears again.

To run SAMPLE from DEBUG, enter:

◊ G

Press the space bar. Ok so far. Now press a non-space key.

Our break point just woke up DEBUG. The registers and single unassembled instruction are displayed.

Enter:

◊ U cs:address

Address is the value of the IP register minus 10 hexadecimal. Since DEBUG is rather primitive, the value of the IP register minus 10 hexadecimal must be calculated by hand. The instruction pointer is pointing one instruction past the instruction that caused the break point. By going back ten hexadecimal instructions, DEBUG should sync up.

The instruction at offset 3BH is:

CS:

MOV BYTE PTR [13],0

Jed says,"There it is! I just knew Jake's helper routines were the problem! His code segment override instruction is writing a zero byte right over my code! Who knows what that's doing!"

Enter:

◊ U 0

Location 13H happens to be the offset of a conditional jump instruction. The relative offset of the conditional jump is being set to zero. If you are an 8086 guru, you obviously know that the JE will ALWAYS fall through if the relative offset is zero. What a subtle BUG!

Now we will take a quick look at how this problem would be solved using SoftICE as a stand-alone debugger. But first we must exit from debug.

Before exiting the debugger, it's always a good idea to disable all the break points, unless ACTION is set to HERE. If you do not do this, when a break point occurs and ACTION tries to return to a debugger that is not loaded, the results are unpredictable. We've changed the ACTION to INT3, so we have to disable the break point.

To bring up the window, press:

◊ ALT D

List the break point by entering:

◊ BL

Notice that the break point description line is highlighted. The highlighted break point is the last break point that occurred.

Notice that the break point number is 0. To disable break point zero, enter:

◊ BD 0

List the break point again by entering:

◊ BL

The asterisk (*) after the break point number shows that the break point is disabled.

To clear the break point, enter:

◊ BC 0 Enter BL again.

Notice that there are no break point lines displayed.

Exit from SoftICE, then exit from the debugger, by entering:

◊ X

◊ Q

The next part of the tutorial demonstrates how SoftICE can be used to find the same problem as a stand-alone debugger. SoftICE will be used as a source level debugger.

To prepare SoftICE to debug at source level it must have been installed in your CONFIG.SYS file, and extended memory allocated for symbols and source files. SoftICE can only be used as a source level debugger if you have extended memory on your system. If you do not have extended memory you may still want to read through the rest of the tutorial to see the capabilities of SoftICE with extended memory. If you have not loaded S-ICE.EXE in your CONFIG.SYS file with memory reserved for symbols, do so at this time.

To debug the sample program with SoftICE as a stand-alone debugger we must use the SoftICE program loader (LDR.EXE). To load the sample program(SAMPLE.EXE), the symbol file (SAMPLE.SYM) and the source file(SAMPLE.ASM) enter:

◊ LDR SAMPLE

You are now in SoftICE with SAMPLE.EXE loaded into memory. Notice that SoftICE occupies the full screen. SoftICE switches to its wide mode whenever a program loaded. The source from SAMPLE.ASM should be visible in the code window. In addition, the register window and the DATA windows are visible.

◊ Step through one instruction by pressing F10.

Notice that the reverse video bar moves to the next instruction to be executed after a program step.

◊ Press F6.

This places the cursor in the code window.

Now experiment with the ?, ?, pageUp, and pageDn keys to move the cursor and scroll the source file.

Move the cursor down to line 42 with the ? key.

◊ Press F9.

We have just set an execution break point on line 42. The line should be highlighted, showing you that a break point has been set on it.

Enter:

◊ BL

This shows the break point that we have just set.

◊ Now press ALT D.

This exits SoftICE, and causes the sample program to execute until it encounters the break point on line 42. SoftICE should immediately come back, with the reverse video bar on line

◊ Press F6 again.

This will bring the cursor back to the command window. Now enter:

◊ BC *

This will clear all the break points (there should only be one set).

◊ Now exit from SoftICE by pressing ALT D.

You are back to the sample program. Type a few keys just to make sure it is still broken.

◊ Now pop SoftICE back up with ALT D.

Since the bug has already occurred, we want to restart the program. Enter:

◊ EXIT RD

This command forces the sample program to exit. The R tells SoftICE to restore the interrupt vectors to the state they were when the sample program was loaded with LDR. The D tells SoftICE to delete any currently pending break points. The R and the D are not necessary in this case, but it is good to get in the habit of specifying them when exiting a program that was loaded with LDR.EXE.

You are now back at the DOS prompt. Reload the program by entering:

◊ LDR SAMPLE.EXE

Notice the suffix.EXE was specified this time. When the suffix is specified, SoftICE does not attempt to load a symbol file or source file. In this case the symbol file and source file are already in memory.

Enter:

◊ SYM

This displays the public symbols of the sample program.

◊ Press Esc to get back to the prompt.

We will now set a range break point similar to the one we set while using SoftICE as an assistant to debug. This time we will use symbols to set the break point. Enter:

◊ BPR START .82 W

This will set a range break point in our code segment from the symbol START to line 82 of the source file.

Enter:

◊ BL

You can verify that the break point has been set properly.

◊ Press ALT D.

◊ Press a non-space key.

We're back in SoftICE. Notice that the current instruction (the line with the reverse video bar) is the instruction after the one that caused the break point.

◊ To see the actual code press the F3 key.

This places SoftICE in mixed mode. Notice that the reverse video bar covers 2 lines. This is the actual code line and the source code line of the current instruction.

◊ Press the F3 key again.

We are now in code mode. No source lines are visible. The instruction above the reverse video bar is the instruction that caused the range break point to go off.

◊ Press the F3 key again to get back to source mode.

Now we will fix the bug in the sample program.

Exit the sample program and go back to the DOS prompt by entering:

◊ EXIT RD

Re-load the sample program by entering: LDR SAMPLE. EXE

Set the code window in code mode by pressing the F3 key twice.

Un-assemble at the broken routine by entering:

◊ U not_space

We will now use the SoftICE interactive assembler to fix the problem.

Enter:

◊ A not_space

SoftICE will prompt you with the address.

Enter:

◊ NOP

Press ENTER to exit from the assembler.

Notice in the code window that there is a NOP instruction in place of the CS over-ride at offset 003BH.

◊ Press the F3 key to get back to source mode, (the source code of course is not modified).

◊ Press ALT D to run the mended sample program.

Enter:

◊ spaces and some non-spaces

It works! You fixed the bug!

To get out of Jed's program, and return to DOS, press:

◊ CTRL C

Now we're going to demonstrate another feature of SoftICE.

Enter:

◊ LDR SAMPLE.EXE

This will load the sample program in one more time.

Enter:

◊ RIP HANG_EXAMPLE

The first two displayed instructions are:

- CLI
- JMP $

Notice that the jump instruction jumps to itself. This infinite loop would normally hang the system in an unrecoverable fashion.

Enter:

◊ BREAK ON

We have just turned on BREAK mode. BREAK mode will cause the system to run slightly slower, but will allow SoftICE to come up even when the system would normal be hung.

◊ Exit from SoftICE by pressing ALT D.

Your system is now hung. For those non-believers, press:

◊ CTRL ALT DEL

Nothing happens! It is definitely hung.

Now press ALT D.

The SoftICE window is back!

To get out of the infinite loop, enter:

◊ EXIT RD

You are now back at DOS. Try a few directories to get a feel for the performance degradation. Many people feel comfortable leaving BREAK ON as a configuration default.

Turn BREAK mode off again by entering:

◊ ALT D BREAK OFF ALT D

Do a few directories to get a comparison of the speed. That's it! Have fun! It's time to start experimenting and debugging on your own. Browse through the rest of the manual and refer to specific sections when necessary.

4 Commands

This chapter contains syntax listings for each SoftICE command, and explanations and examples for each command. All numbers are in hexadecimal; any number can be an expression using +,-,/,*, or registers. All commands are case-insensitive. Words that are in italics the command syntax statements must be replaced by an actual value, rather than typing in the italicized word.

The following notational conventions are used throughout this section

[ ]

Brackets enclose an optional syntax item.

< >

Angle brackets enclose a list of items or choices.

x | y

Vertical bars separate alternatives. Use either item x or item y.

count

Count is a byte value that specifies the number of times break point conditions must be met before the actual break point occurs. If no count is specified, the default value is 1. Each time the SoftICE window is brought up, the counts are reset to the values originally specified.

verb

Verb is a value that specifies what type access the break point will apply to. It can be set to 'R' for reads, 'W' for write 'RW' for reads and writes, or 'X' for execute.

address

Address is a value that is made of two 16-bit words, separated by a colon. The first word is the segment address, and the second word is the segment offset. The addresses can be constructed of registers expressions, and symbols. Thee address may also contain the special characters "$", ".", and "@". See section 3-8 (Command Syntax) for a description of these special characters.

break-number

Break-number is an identification number that identifies the break point to use when you are manipulating break points (e.g., editing, deleting, enabling, or disabling them). The break-number can be a hexadecimal digit from 0 to F.

list

List is a series of break-numbers separated by commas or spaces.

mask

Mask is a bitmask that is represented as: combination of 1's, 0's, and X's. X's are don't-care bits.

Example

```
BPIO 21 W EQ M 1XXX XXXX
```

This command will cause a break point to occur if port 21H is written to with the high order bit set.

GT, LT

GT and LT are command qualifiers that unsigned comparisons of values.

Using Break Point Commands

Introduction

SoftICE has break point capability that has traditionally only been available with hardware debuggers. The power and flexibility of the 80386 chip allows advanced break point capability without additional hardware.

All of SoftICE's break points are sticky. That means they don't disappear automatically after they've been used; you must intentionally clear or disable them using the BC or the BD commands. SoftICE can handle 16 break points at one time. You can have up to ten break points of a single type except for break points on memory location (BPMs), of which you can only have four, due to restrictions of the 80386 processor.

Break points can be specified with a count parameter. The count parameter tells SoftICE how many times the break point should be ignored before the break point action occurs.

Setting Break Points Commands

Break points can be set on memory location reads and writes, memory range reads and writes, program execution and port accesses. SoftICE assigns a one-digit hexadecimal number (0-F) to each break point. This break-number is used to identify break points when you set delete, disable, enable, or edit them. The break point setting commands are:

BPM, BPMB, BPMW, BPMD	Set break point on memory access or execution
BPR	Set break point on memory range
BPIO	Set break point on I/O port access
BPINT	Set break point on interrupt
BPX	Set/clear break point on execution
CSIP	Set CS:IP range qualifier
BPAND	Wait for multiple break points to occur

BPM, BPMB, BPMW, BPMD

Set break point on memory access or execution

Syntax

```
BPM[size]address[verb][qualifier value][C=count]
```

size

Size is actually a range covered by this breakpoint. For example, if you use double word, and the third byte of the dword is modified, a breakpoint occurs. The size is also important if you specify the optional qualifier.

Value	Description
B	Byte
W	Word
D	Double Word

verb

R, W, RW, or X

qualifier

EQ, NE, GT, LT, M

EQ	Equal
NE	Not Equal
GT	Greater than
LT	Less Than
M	Mask

These qualifiers are only applicable to the read and write break points. value -- A byte, word, or double word value, depending on the size specified.

Comments:

The BPM commands allow you to set a break point on memory reads or writes or execution.

If a verb is not specified, RW is the default.

If a size is not specified, byte is the default.

All of the verb types except X cause the program to execute the instruction that caused the break point. The current CS:IP will be the instruction after the break point. If the verb type is X, the current CS:IP will be the instruction where the break point was set.

If R is specified, then the break point will occur on read access and on write operations that do not change the value of the memory location.

If the verb type is R, W or RW, executing an instruction at the specified address will not cause the break point action to occur.

Note

If BPMW is used, the specified address must start on a word boundary. If BPMD is used, the specified address must point to a double word boundary.

Example

```
BPM 1234:SI W EQ 10 C=3
```

This command defines a break point on memory byte access. The third time that 10 hexadecimal is written to location 1234:SI, the break point action will occur. BPM CS:1235 X This command defines a break point on execution. The break point action will occur the first time that theinstruction at address CS:1235 is reached. The current CS:IP will be the instruction where the break point was set.

```
BPMW DS:FOO W EQ M 0XXX XXXX XXXX XXX1
```

This command defines a word break point on memory write. The break point action will occur the first time that location DS:FOO has a value written to it that sets the high order bit to 0 and the low order bit to 1. The other bits can be any value.

```
BPM DS:1000 W GT 5
```

This command defines a byte break point on memory write. The break point action will occur the first time that location DS:1000 has a value written to it that is greater than 5.

BPR

BPR

Set break point on memory range

Syntax

```
BPR start-address end-address [verb] [C=count]
```

start-address, end-address

specifies a memory range

verb

R, W, RW, T or TW

Comments

The BPR command allows you to set a break point across a range of memory.

All of the verb types except T or TW cause the program to execute the instruction that caused the break point. The current CS:IP will be the instruction after the break point.

There is no range break point on execution. If a range break point is desired on execution, R must be used. An instruction fetch is considered a read for range break points.

If a verb is not specified, W is the default.

The range break point will degrade system performance in certain circumstances. Any read or write within the 4K page that contains the break point range is analyzed by SoftICE. This performance degradation is usually not noticeable, however, degradation could be extreme in exception cases.

The T and TW verbs enable back trace ranges on the specified range. They do not cause break points, but instead log instruction information that can be displayed later with the SHOW or TRACE commands. For more information on back trace ranges, see chapter 9.

Example

```
BPR B000:0 B000:1000 W
```

This command defines a break point on memory range. The break point will occur if there are any writes to the monochrome adapter video memory region.

BPIO

BPIO

Set break point on I/O port access

Syntax

`BPIO port [verb] [qualifier value] [C=count]`

port

A byte or word value

verb

R, W, or RW

qualifier

EQ, NE, GT, LT, M

`EQ`	Equal
`NE`	Not Equal
`GT`	Greater than
`LT`	Less Than
`M`	Mask

Comments

The BPIO command allows you to set a break point on I/O port reads or writes.

If value is specified, it is compared with the actual data value read or written by the IN or OUT instruction causing the break point. The value may be a byte or a word. If the I/O is to a byte port, then the lower 8 bits are used in the comparison.

The instruction pointer (CS:IP) will point to the instruction after the IN or OUT instruction that caused the break point.

If a verb is not specified, RW is the default.

Example

`BPIO 21 W NE FF`

This command defines a break point on I/O port access. The break point will occur if the interrupt controller one mask register is written with a value other than FFH.

BPIO 3FE R EQ M 11XX XXXX

This command defines a byte break point on I/O port read. The break point action will occur the first time that I/0 port 3FE is read with a value that has the two high order bits set to 1. The other bits can be any value.

BPINT

BPINT

Set break point on interrupt

Syntax

`BPINT int-number [ < AL | AH | AX >= value] [C = count]`

int-number

Interrupt number from 0 - FF hex

value

A byte or a word value

Comments

The BPINT command allows breaking on the execution of a hardware or a software interrupt. By optionally qualifying the AX register with a value, specific DOS or BIOS calls can be easily isolated.

If no value is specified, a break point will occur when the interrupt specified by int-number occurs. This interrupt can be a hardware, software, or internal interrupt.

The optional value is compared with the specified register (AH, AL, or AX) when the interrupt occurs. If the value matches the specified register, then the break point will occur.

When the break point occurs, if the interrupt was a hardware interrupt, the instruction pointer (CS:IP) will point to the first instruction within the interrupt routine. The INT? command can be used to see where execution was when the interrupt occurred. If the interrupt was a software interrupt, when the break point occurs, the instruction pointer (CS:IP) will point to the INT instruction causing the interrupt.

Example

`BPINT 21 AH=4C`

This command defines a break point on interrupt 21H The break point will occur when DOS function call 4CH (terminate program) is called.

BPX

BPX

Set/clear break point on execution

Syntax

```
BX [address] [C=count]
```

Comments

The BPX command allows you to set or clear a point-and-shoot execution break point in source. When the cursor is in the code window the address is not required. The execution break point is set at the address of the current cursor location. If an execution break point has already been set at the address of the current cursor location, then the break point is cleared.

If the code window is not visible or the cursor is not in the code window then the address must be specified. If an offset only is specified then the current CS register value used as the segment.

Technical Note

BPX uses an interrupt 3 style of break point unless the specified address is ROM. This is used instead of a break point register to make more execution break points available. If your circumstances require the use of a break point register for some reason (code not loaded yet for example) you can set an execution break point with the BPM command.

Example

```
BPX.1234
```

This sets an execution break point at source line 1234.

CSIP

CSIP

Set CS:IP range qualifier

Syntax

```
CSIP [OFF | [NOT] start-address end-address]
```

NOT

When NOT is specified, the break point will only occur if the CS:IP pointer is outside the specified range.

OFF

Turns off CS:IP checking

Comments

The CSIP command causes a break point to be dependent upon the location of the instruction pointer when the break point conditions are met. This function is often useful when a program is suspected of accidentally modifying code outside of its boundaries.

When break point conditions are met, the CS:IP registers are compared with a specified range. If they are within the range, the break point is activated. To activate the break point when CS:IP is outside the range, use the NOT parameter.

When a CSIP range is specified, it applies to ALL break points that are currently active.

If no parameters are specified, the current CSIP range is displayed.

Example

```
CSIP NOT F000:0 FFFF:0
```

This command causes the break points to occur only the CS:IP is NOT in the ROM BIOS when the break point conditions are met.

BPAND

BPAND

Wait for multiple break points to occur

Syntax

`BPAND list | * | OFF`

list

A series of break-numbers separated by commas or spaces

`*`

ANDs together all break points

Comments

The BPAND command does a logical AND of two or more break points, activating the break point only when conditions for all break points are met.

Sometimes conditions arise when you don't want a break point to occur until several different conditions are met. The BPAND command allows specifying two or more break points that must occur before the action is generated. This function allows more complex break point conditions to be set.

Each time the BPAND command is used, the specified break point numbers are added to the list until BPAND OFF is used.

You can tell which of the break-numbers are ANDed together by listing the break points with the BL command. The break points that are ANDed together will have an ampersand (&) after their break-number.

Once break points have been ANDed together, each remains ANDed until it is cleared, or until BPAND is turned off.

Example

`BPAND 0,2,3`

This command causes the conditions of the break points 0, 2, and 3 to be logically tied together. The break occurs only when the conditions of all three are met. For example, if the conditions of break points 2 and 3 have both been met at least once, but the conditions of break point 0 have not been met at all yet, then the action will not occur until break point 0 conditions are met.

Manipulating Break Points Commands

SoftICE provides several commands for manipulating break points. Manipulation commands allow listing, modifying, deleting, enabling, and disabling of break points. Break points are identified by break-numbers which are hexadecimal digits from 0 to F. The break point manipulation commands are:

BD	Disable break points
BE	Enable break points
BL	List break points
BPE	Edit break point
BPT	Use break point as a template
BC	Clear break points
WATCH	Set watch expressions
CWATCH	Clear watch expressions

BD

BD

Disable break points

Syntax

```
BD list | *
```

list

A series of break-numbers separated by commas or spaces

*****

Disables all break points

Comments

The BD command is used to temporarily deactivate break points. The break points can be reactivated with the BE (Enable break points) command.

You can tell which of the break-numbers are disabled by listing the break points with the BL command. The break points that are disabled will have an asterisk (*) after their break-number.

Example

```
BD 1,3
```

This command temporarily disables break points 1 and 3.

BE

BE

Enable break points

Syntax

```
BE list | *
```

list

Aseries of break-numbers separated by commas or spaces

*****

Enables all break points

Comments

The BE command is used to reactivate break points that were deactivated by the BD (Disable break points) command.

Note that a break point is automatically enabled when it first defined.

Example

```
BE 3
```

This command enables break point 3.

BL

BL

List break points

Syntax

BL

Comments

The BL command displays all break points that are currently set. For each break point, BL lists the break-number, break point conditions, break point state, and count.

The state of a break point is either enabled or disabled. If the break point is disabled, an asterisk (*) is displayed after its break-number. If an enabled break point was used in a BPAND command, an ampersand (&) is displayed after its break-number. The break point that most recently caused an action to occur is highlighted.

The BL command has no parameters.

Example: BL

This command displays all the break points that have been defined. A sample display, which shows four break points, follows:

```
0) BPMB 1234:0000 W EQ 0010 C=03
1) *BPR B000:0000 B000:1000 W C=01
2) BPIO 0021 W NE 00FF C=01
3) BPINT 21 AH=4C C=01
```

Note that in this example, break point 1 is preceded with an asterisk (*), showing that it has been disabled.

BPE

BPE

Edit break point

Syntax

```
BE break-number
```

Comments

The BPE command loads the break point description into the edit line for modification. The command can then be edited using the editing keys, and re-entered by pressing the ENTER . This command offers a quick way to modify the parameters of an existing break point.

Example

```
BPE 1
```

This command moves a description of break point 1 into the edit line and removes break point 1. Pressing the ENTER key will cause the break point to be re-entered.

BPT

BPT

Use break point as a template

Syntax

```
BT break-number
```

Comments

The BPT command uses an existing break point description as a template for a new break point.

A description of the existing break point is loaded into the edit line. The break point referenced by break-number is not altered. This command offers a quick way to create a new break point that is similar to an existing break point.

Example

```
BPT 3
```

This command moves a template of break point 3 into the edit line. When the ENTER key is pressed, a new break point is added.

BC

BC

Clear break points

Syntax

```
BC list | *
```

list

A series of break-numbers separated by commas or spaces

*****

Clears all break points

Comments

The BC command is used to permanently delete one or more break points.

Example

```
BC *
```

This command clears all break points.

WATCH

WATCH

The WATCH commands are used to display the results of expressions.

Syntax

```
WATCH [size] expression
```

size

B - display a byte in hexadecimal format

W - display a word in hexadecimal format

D - display a dword in hexadecimal pointer format

I - display a word in decimal (int) format

E - display a dword in decimal (long int) format

U - display a word in unsigned decimal format

F - display a dword in unsigned decimal format

S - display a 4-byte (float) FP real

L - display a 8-byte (double) FP real

A - display an ASCII string (33 char. max.)

expression

The results of expression are displayed in the format of the size specified. If no size is specified, byte will be assumed. The expressions being watched are displayed in the watch windows. There can be up to eight watch expressions at a time. Every time SoftICE screen is popped up, the watch window will display the expressions' current values.

Comments

Each line in the watch windows contains the following information:

- A watch number from 0 to 7.
- The hexadecimal address of the expression.
- The current value of the expression displayed in the appropriate format.
- The expression being evaluated.

Example

```
WATCH W FooVariable
```

CWATCH

CWATCH

The CWATCH command is used to clear one or watch expressions from the watch window.

Syntax

```
CWATCH list | *
```

list

This is a list of watch numbers from 0-7 separated by commas. Watch-numbers are the numbers displayed on the beginning of each line in the watch window.

*****

Clears all watch expressions

Comments

After clearing the expressions, the ones still remaining in the window are renumbered sequentially starting at 0. If there are no more watch expressions, the window disappears. When using CWATCH with a list be careful. If your list was 1,3, this would actually remove the first and fourth watch in your current watch window. When the first watch is removed, all remaining watches are renumbered beginning at 1. So the actual list would be 1,2.

Example

```
CWATCH 1,2
```

Using Other Commands

Display and Edit Commands

U	Unassemble instructions or display source
R	Display or change registers
MAP	Display system memory map
D	Display memory in the most recently specified format
DB	Display memory in byte format
DW	Display memory in word format
DD	Display memory in double word format
E	Edit memory in the most recently specified format
EB	Edit memory bytes
EW	Edit memory words
ED	Edit memory double words
ES	Edit stack value
INT?	Display last interrupt number
? or H	Display help information
VER	Display SoftICE version number

U

U

Unassemble instructions or display source

Syntax

`U [address] [L[=]length]`

length

The number of instructions to be unassembled

Comments

The U command displays the instructions of the program being debugged.

If length is not specified, the length defaults to eight lines if available, or one less than the screen length.

If address is not specified, the command unassembles at address starting at the first byte after the last byte unassembled by a previous unassemble command. If the has been no previous unassemble command, the address defaults to the current CS:IP.

If the code window is visible, the instructions are displayed in the code window.

If source is loaded for the address range specified then source lines may be displayed depending on the current source mode.

Example

`U $-10`

This command unassembles instructions beginning 10 hexadecimal bytes before the current address.

`U .499`

This command displays the current source file starting at line 499. The code window must be visible and in source mode.

R

R

Display or change registers

Syntax

```
R register-name [ [ = ]value] ]
```

register-name

Any of the following: EAX, AX, AH, AL, EBX, BX, BH, BL, ECX, CX, CH, CL, EDX, DX, DH, DL, EDI, DI, ESI, SI, EBP, BP, ESP, SP, CS, DS, ES, SS, FS, GS, FL

value

If register-name is any name other than FL, value is a hex value or an expression. If register-name is FL, value is a series of one or more of the following flag symbols, each optionally preceded by a plus or minus sign:

`O`	Overflow flag
`D`	Direction flag
`I`	Interrupt flag
`S`	Sign flag
`Z`	Zero flag
`A`	Auxiliary carry flag
`P`	Parity flag
`C`	Carry flag

Comments

The R command displays or changes register values.

If no parameters are supplied, all register and flag value are displayed, as well as the instruction at the current CS:IP address.

If both register-name and value are supplied, the specified register's contents are changed to the value.

To change a flag value, use FL as the register-name, followed by the symbols of the flag whose values you want to toggle. To turn a flag on, precede the flag symbol with a plus sign. To turn a flag off, precede the flag symbol with a minus sign. The flags can be listed in any order.

Examples

RAH 5

This command sets the AH register equal to 5.

R FL = OZP

This command toggles the O, Z, and P flag values.

R FL

This command displays the current flag values, and allows them to be changed.

RFL O + A-C

This command toggles the O flag value, turns on the flag value, and turns off the C flag value.

MAP

MAP

Display system memory map

Syntax

MAP

Comments:

The MAP command displays the names, locations, and sizes of system memory components. The size is displayed in paragraphs. One paragraph is equivalent to 10 hexadecimal bytes.

The component that the CS:IP register currently points to is highlighted.

Use the MAP command when:

- A break point occurs and CS:IP is not in a known memory region.
- You want to get control within a resident program or system program. A range break point can be set based on the starting address and size reflected by MAP.
- You suspect a program or system component of writing over code outside of its memory space. MAP is used to obtain the memory address of the region to use with the CSIP command.
- You need to find out which resident program owns certain interrupt vectors.

Example

MAP

The following is a sample display produced by the MAP command:

```
Start      Length
0000:0000  0040 Interrupt Vector Table
0040:0000  0030 ROM BIOSVariables
0070:0000  00FE I/O System
016E:0000  06B7 DOS
0842:0000  02CE DOS File Table & Buffers
A000:0000  5E00 System BUS
F000:0000  1000 ROM BIOS
```

Versions of DOS lower than 3.1 display program addresses instead of displaying the program names.

D, DB, DW, DD

D, DB, DW, DD

Display memory

Syntax

```
D [size] [address] [L[ = ]length]
```

size

B	Byte
W	Word
D	Double Word

length

The number of bytes to be displayed.

Comments

The D command displays the memory contents of the specified address.

The contents are displayed in the format of the size specified. If no size is specified, the last size used will be displayed. The ASCII representation is also displayed for all forms.

If address is not specified, the command displays memory at the address starting at the first byte after the last byte displayed.

If length is not specified, it defaults to eight lines, or fewer if the window is smaller.

If the data window is visible, the data is displayed in the data window and the length is ignored.

Example

```
DW DS:00 L=8
```

This command displays, in word format and in ASCII format, the value of the first eight bytes of the current data segment.

E, EB, EW, ED

E, EB, EW, ED

Edit memory

Syntax

`E [size ] address [data-list]`

size

B	Byte
W	Word
D	Double Word

data-list

List of data objects of the specified size (Bytes, Words or Double Words) or quoted strings separated by commas or spaces. The quoted string can begin with a single quote or a double quote.

Comments

The E commands display the memory contents at the specified address, and allow you to edit the values.

These commands display the memory contents in ASCII format, and in the format of the size specified.

A memory editor is provided for quick memory updates. Memory can be edited by typing ASCII characters, or by typing byte, word, or double word values. If no size is specified, the last size used will be assumed. The memory Editing key strokes are:

UP ARROW	Move cursor up
DOWN ARROW	Move cursor down
RIGHT ARROW	Move cursor right
LEFT ARROW	Move cursor left
SPACE	Move cursor to next element
TAB	Toggle between numeric and ASCII areas
ESC or ENTER	Exit memory editor

As values are input, the actual memory locations are updated. All numeric values are hex numbers. To toggle between the ASCII and numeric display areas, press the TAB key.

If the data window is visible, the data is edited in the data window, otherwise the data is edited in the command window.

The data display length defaults to 8 lines if in the command window, or to the size of the data window if it's visible.

If no parameters are supplied, the cursor moves into the data window if the data window if visible. If the data window is not visible, the data is edited in the command window at the last address displayed or edited.

Examples

`EB 1000:0`

This command displays, in byte format, up to six lines containing both the numeric and the ASCII representation of the values of the data starting at location 1000:0000. Once the lines are displayed, you can edit the values.

`EB 8000:0 “Hello”,0D`

This command replaces the values starting at locatio 8000:0000 with the string “Hello” followed by a carriage return.

ES

ES

The ES command changes the value of the highlighted entry in the stack window to the value of hex value.

Syntax

`ES [hex value]`

hex value

4 digit value

Example

`ES 45B9`

Changes the value of the highlighted entry in the stack window to 45B9.

INT?

INT?

Display last interrupt number

Syntax

INT?

Comments

The INT? command displays the address and the number the last interrupt that happened.

Example

INT?

An example of the display produced by the INT? command follows:

```
Last Interrupt: 16
At: 0070:0255
```

This example shows that the last interrupt generated in the system before the SoftICE window was brought up was an interrupt 16 hexadecimal, at location 0070:0255H. If the last interrupt that happened was a software interrupt, unassembling the code at 0070:0255H will show the interrupt instruction. If it was a hardware interrupt, unassembling the code will show the instruction that was executing when the hardware interrupt occurred.

? or H

? or H

Display help information

Syntax

```
< ? | H > [command | expression]
```

Comments

The ? command and the H command both display help information.

If no parameters are specified, help displays short descriptions of all the commands and operators, one screen at a time. Press any key to continue, or press ESC to quit displaying help.

If command is specified, help displays more detailed information on the specified command, including the command syntax and an example.

If expression is specified, the expression is evaluated and the result is displayed in hexadecimal, decimal, and ASCII.

Examples

```
? ALTKEY
```

This command displays information about the ALTKEY command, including its syntax and an example.

```
H 10 + 14*2
```

This command displays: 0038 00056 “8”. These are the hexadecimal, decimal and ASCII representations of value of the expression “10 + 14*2”.

VER

VER

Display SoftICE version number

Syntax

VER

Example

VER

This command displays the SoftICE version and the Nu-Mega Technologies copyright message.

I/O Port Commands

I or IB	Input from byte I/O port
IW	Input from word I/O port
O or OB	Output to byte I/O port
OW	Output to word I/O port

I, IB, IW

I, IB, IW

Input from I/O port

Syntax

`I [size] port`

Size

`B`	Byte
`W`	Word

port

A byte or word value

Comments

The input from port commands are used to read and display a value from a hardware port. Input can be done From byte or word ports. If no size is specified, the default is byte.

Example

`I 21`

This command displays the mask register for interrupt controller one.

O, OB, OW

O, OB, OW,

Output to I/O port

Syntax

`O [size] port value`

size

`B`	Byte
`W`	Word

port

A byte or word value

value

A byte for a byte port or a word for a word port

Comments

The output to port commands are used to write a value to a hardware port. Output can be done to byte or word ports If no size is specified, the default is byte.

Example

`O 21 FF`

This command masks off all the interrupts for interrupt controller one.

Transfer Control Commands

X	Exit from SoftICE window
G	Go to address
T	Trace one instruction
P	Program step
HERE	Go to current cursor line
GENINT	Force an interrupt
EXIT	Force exit of current DOS program
BOOT	System boot (retain SoftICE)
HBOOT	Hard system boot (total reset)

X

X

Exit from SoftICE window

Syntax

x

Comments

The X command exits the SoftICE window and restores control to the program that was interrupted to bring up SoftICE. The SoftICE window disappears. If any break points have been set, they become active.

Example

x

G

G

Go to address

Syntax

```
G [=start-address] [break-address]
```

Comments

The G command exits from the SoftICE window with a single one-time execution break point set. In addition, all sticky break points are armed.

Execution begins at the current CS:IP unless the start-address parameter is supplied. In that case execution begins at start-address. Execution continues until break-address is encountered, the window pop-up key sequence is used, or a sticky break point occurs.

The break-address must be the first byte of an instruction opcode.

When the specified break-address is reached, the current CS:IP will be the instruction where the break point was set.

The G command with no parameters behaves the same as the X command.

The non-sticky execution break point uses an 80386 break point register, unless all break point registers have been allocated to sticky break points. In that case, an INT 3 style break point is implemented. When this case occurs, the G and P commands will not work correctly in ROM. An error message will be displayed if this is attempted.

Example

```
G CS:1234
```

This command sets a one time break point at CS:1234 94

T

T

Trace one instruction

Syntax

```
T [=start-address] [count]
```

Comments

The T command single steps one instruction by utilizing the single step flag.

Execution begins at the current CS:IP unless the start-address parameter is specified. If start-address is specified, CS:IP is changed to start- address prior to single stepping.

If count is specified then SoftICE single steps count time The TRACE command will continue until the count is exhausted or the Esc key is pressed, regardless of which break points are reached.

In source mode, the T command steps to the next source statement. If the current statement is a procedure or function call, and source exists for the routine being called, T steps into the call. If there is no source available for the called procedure or function, T steps over the routine.

Example

```
T = 1284 3
```

This command single steps through three instruction starting at memory location 1284.

P

P

Program step

Syntax

P[RET]

Comments

The P command is a logical program step. One instruction at the current CS:IP is executed unless the instruction is a call, interrupt, loop, or repeated string instruction. In those cases, the entire routine or iteration is completed before control is returned to SoftICE.

The P command uses a one-time execution break point. The non-sticky execution break point uses an 80386 break point register, unless all break point registers have been allocated to sticky break points. In that case, an INT3 style break point is implemented. When this case occurs, the P and G commands will not work correctly in ROM. An error message will be displayed if this is attempted.

In source mode, the P command steps to the next source statement. If the current statement is a procedure or function call, the P command steps over the it.

If RET is specified, the P command will step until the next RET or IRET instruction.

Example

P

This command executes one ‘program step’.

HERE

HERE

Go to current cursor line

Syntax

HERE

Comments

The HERE command executes until the program reaches the current cursor line. HERE is only available when the cursor is in the code window. If the code window is not visible or the cursor is not in the code window, use the G command instead.

The HERE command exits from SoftICE with a single one-time execution break point set. In addition, all sticky break points are armed.

Execution begins at the current CS:IP and continues until address of the current cursor position in the code window encountered, the window pop-up key sequence is used, a sticky break point occurs.

The non-sticky execution break point uses an 80386 break point register, unless all break point registers have been allocated to sticky break points. In that case, an INT 3 style break point is implemented. When this case occurs, the HERE command will not work correctly in ROM. An error message will be displayed if this is attempted.

Example

HERE

This example sets an execution break point at the current cursor position, then exits from SoftICE and begins execution at the current CS:IP.

Default Function Key

F7

GENINT

GENINT

Force an interrupt

Syntax

GENINT INT1 | INT3 | NMI | interrupt-number

interrupt-number

a number in the range 00 - FF

Comments

The GENINT command forces an interrupt to occur. This function can be used to hand off control to another debugger when using SoftICE with another software debugger. It can also be used to test interrupt routines.

The GENINT command simulates the processing sequence of a hardware interrupt or an INT instruction. It pushes the flags, the CS register, and the IP register, then changes the value of the CS and IP registers to the value of the interrupt vector table entry corresponding with the specified interrupt number.

Example

```
GENINT NMI
```

This forces a non-maskable interrupt. This will give control back to CodeView if SoftICE is being used as an assistant to CodeView.

EXIT

EXIT

Force exit of current DOS program

Syntax

`EXIT [R][D]`

R

Restore the interrupt vector table

D

Delete all break points

Comments

The EXIT command attempts to abort the current program by forcing a DOS exit function (INT 21H, function 4CH) This command will only work if the DOS is in a state where it is able to accept the exit function call. If this call is made from certain interrupt routines, or other times when the DOS is not ready, the system may behave unpredictably.

This function does NOT do any system resetting other than the interrupt table when the R option is used. This means that BIOS variables, video modes and other systems level data are not restored.

Using the R option will cause the interrupt vectors to be restored to whatever they were the last time they were saved. SoftICE saves the interrupt vectors when it is loaded, when a program is loaded with LDR.EXE, and when the VECS S command is used.

Note

To re-start a program that has been loaded with the SoftICE program loader (LDR.EXE) do the following:

```
EXIT R
LDR prog.EXE
```

The EXIT command will restore the interrupt table to the values it contained before the program was loaded, then exit to the command processor. By running the LDR utility and specifying the .EXE suffix, the program is loaded back in without re-loading symbols and source. The symbols and source will remain in memory.

Caution

The EXIT command should be used with care. Since SoftICE can be popped up at any time, a situation can occur where the DOS is not in a state to accept an exit function call. Also, the EXIT command does not do any program specific resetting. For instance, the EXIT command does not reset the video mode. If your program has placed the video BIOS and hardware in a particular video mode, it will stay in that mode after the EXIT command.

Example

```
EXIT R
```

Restores the interrupt table and exits the current program. The R option should be used if exiting from a program loaded with the SoftICE program loader LDR.EXE.

BOOT

BOOT

System boot (retain SoftICE)

Syntax

BOOT

Comments

The BOOT command resets the system and retains SoftICE. BOOT is required to debug boot sequences, DOS loadable drivers, and non-DOS operating systems.

BOOT is implemented with an Interrupt 19H ROM BIOS call. In some instances memory may be corrupted to the point where Interrupt 19 will not work. If this occurs, bring up SoftICE and use the HBOOT command.

For BOOT to work properly, SoftICE should be installed as a loadable driver in CONFIG.SYS before any other device drivers. This is so SoftICE can restore the original system state as accurately as possible.

Example

BOOT

This command makes the system reboot. SoftICE remains resident.

HBOOT

HBOOT

Hard system boot (total reset)

Syntax

HBOOT

Comments

The HBOOT command resets the entire system. SoftICE is not retained in the reset process. HBOOT is sufficient unless an adapter card requires a power-on reset. In those rare cases, the machine power must be recycled.

Example

HBOOT

This command makes the system reboot. SoftICE must be reloaded.

Debug Mode Commands

ACTION	Set action after break point is reached
WARN	Set DOS/ROM BIOS re-entrancy warning mode
BREAK	Break out any time
I3HERE	Direct Interrupt 3's to SoftICE

ACTION

ACTION

Set action after break point is reached

Syntax

```
ACTION [INT1 | INT3 | NMI | HERE | int-number]
```

int-number

Any valid interrupt number (0-FFH). Use this option only if a user-supplied break point qualification routine has taken over that interrupt vector (see section 11.2).

Comments

The ACTION command determines where control is given when break point conditions have been met. In most cases, the desired action is INT3 or HERE, INT3 is typically used if SoftICE is being used with a host debugger, HERE is used when it is desired to return to SoftICE when break point conditions have been met, INT1 and NMI are alternatives for certain debuggers that will not work with the INT3 option. For instance, CODEVIEW works best with ACTION set to NMI.

Use int-number if there is a user-supplied break point qualification routine installed. Using int-number without having a user-supplied break point qualification routine installed causes an error. For more information, see section 11.2,'User-Qualified Break Points'.

If no parameter is supplied with the ACTION command, the current action is displayed.

The default action is HERE.

Example

```
ACTION HERE
```

This command specifies that control will return to SoftICE when break point conditions have been met.

WARN

WARN

Set DOS/ROM BIOS re-entrancy warning mode

Syntax

```
WARN [ON | OFF]
```

Comments

The WARN command is provided for using SoftICE with debuggers that use DOS and ROM BIOS. Many debuggers use DOS and ROM BIOS for screen output and for receiving keystrokes. Since DOS and ROM BIOS are not fully re- entrant, these debuggers may not work properly if break point occurs while the DOS or ROM BIOS is executing.

If WARN ON is set, and ACTION is not HERE, then control will come to Soft- ICE before the actual action occurs. The system displays the current CS:IP and gives you the choice of continuing or returning to SoftICE. Generally, you should choose to return to SoftICE to continue your debugging. Only continue with the host debugger if you know your debugger will not cause DOS or ROM BIOS to be re-entered.

WARN mode should be turned on to use SoftICE with DEBUG, SYMDEB, and CODEVIEW.

If no parameter is specified, the current state of WARN is displayed.

The default is WARN mode OFF.

Example

```
WARN ON
```

This command turns on DOS/ROM BIOS re-entrancy warning mode.

BREAK

BREAK

Break out any time

Syntax

```
BREAK [ON | OFF]
```

Comments

The BREAK command allows popping up the SoftICE window when the system is hung with interrupts disabled. Break mode can be used for the entire debugging session, or it can be turned on and off when it is required.

Break mode degrades system performance slightly. This performance degradation must be weighed against the necessity of breaking out of a hung program. A user may want to have break mode on all the time, even though performance is degraded, because the program could hang at any time.

Unlike other debuggers that can also be brought up at any time, SoftICE does not require an external switch. When BREAK is on, the SoftICE window can be brought up at any time by pressing the current key sequence.

If no parameter is specified, the current state of BREAK is displayed.

The default is BREAK mode OFF.

Example

```
BREAK ON
```

This command turns on break mode. This means that the SoftICE window can be brought up at any time, even if interrupts are disabled.

I3 HERE

I3HERE

Direct Interrupt 3's to SoftICE

Syntax

```
I3HERE [ON | OFF]
```

Comments

The 13HERE command lets you specify that any Interrupt 3 will bring up the SoftICE window. This feature is useful for stopping your program in a specific location.

To use this feature, place an INT 3 into your code at the location where you want to stop. When the INT 3 occurs, it will bring up the SoftICE window. At this point, you can use the R IP command to change your instruction pointer to the instruction after the INT 3, then you can continue debugging.

If no parameter is specified, the current state of 13HERE is displayed.

The default is 13HERE mode OFF.

Example

```
I3HERE ON
```

This command turns on 13HERE mode. Any INT 3's generated after this point will bring up the SoftICE window.

Utility Commands

A	Assemble code
S	Search for data
F	Fill memory with data
M	Move data
C	Compare two data blocks
SERIAL	Redirect console to serial terminal

A

A

Assemble code

Syntax

```
A [address]
```

Comments

The SoftICE assembler allows you to assemble instructions directly into memory. The assembler supports the basic 8086 instruction set with the 80186 and 80286 real address mode extensions. Numeric co-processor instructions and 80386 specific instructions, registers and addressing modes can NOT be assembled.

The A command enters the SoftICE interactive assembler. An address is displayed as a prompt for each assembly line After an assembly language instruction is typed in and ENTER is pressed, the instructions are assembled into memory at the specified address. Instructions must be entered with standard Intel format. Press ENTER at an address prompt to exit assembler mode.

If the address range in which you are assembling instructions is visible in the code window, the instructions will change interactively as you assemble.

The SoftICE assembler supports the standard 8086 family mnemonics, however there are some special additions

- The DB mnemonic is used to define bytes of data directly into memory. The DB command is followed by a list of bytes and/or quoted strings separated by spaces or commas.
- The RETF mnemonic represents a far return.
- WORD PTR and BYTE PTR are used to determine data size if there is no register argument, for example: MOV BYTE PTR ES:[1234],1.
- Use FAR and NEAR to explicitly assemble far and near jumps and calls. If FAR or NEAR is not specified then all jumps and calls are near.
- Operands referring to memory locations should placed in square brackets, for example: MOV AX,[1234].

Example

```
A CS:1234
```

This command prompts you for assembly instruction then assembles them beginning at offset 1234H with the current code segment. Press ENTER at the address prompt after entering the last instruction.

S

S

Search for data

Syntax

```
S address L length data-list
```

data-list

list of bytes or quoted strings separated by commas or spaces. .A quoted string can begin with a single quote or a double quote.

length

length in bytes

Comments:

The S command searches memory for a series of bytes or characters that matches the data-list. The search begins at the specified address and continues for the length specified. The address of each occurrence found in the range is displayed.

Example

```
S DS:SI+10 L CX ‘Hello’,12,34
```

This command searches for the string ‘Hello’ followed by the bytes 12H and 34H starting at offset SI+10 in the current data segment and ending CX bytes later.

F

F

Fill memory with data

Syntax

```
F address L length data-list
```

data-list

list of bytes or quoted strings separated by commas or spaces. A quoted string can begin with a single quote or a double quote.

length

length in bytes

Comments

The F command fills memory with the series of bytes or characters specified in the data-list. Memory is filled starting at the specified address and continuing for the specified length, repeating the data-list if necessary.

Example

F 8000:0 L 100 ‘Test’ This command fills memory starting at 8000:0 for a length of 100H bytes with the string ‘Test’. The string ‘Test’ is repeated until the fill length is exhausted.

M

M

Move data

Syntax

```
M start-address L length end-address
```

length

length in bytes

Comments

The M command moves the specified number of bytes from the start-address in memory to the end-address in memory.

Example

```
M 1000:0 L 200 2000:0
```

This command moves 200H bytes from memory location 1000:0 to memory location 2000:0.

C

C

Compare two data blocks

Syntax

```
C address1 L length address2
```

length

length in bytes

Comments

The C command compares the memory block specified by address1 and the length with the memory block specified address2 and the length. When a byte from the first data block does not match a byte from the second data block, both bytes are displayed, along with their addresses.

Example

```
C 5000:100 L 10 6000:100
```

This command compares the 10H bytes starting at memory location 5000:100 with the 10H bytes starting at memory location 6000:100.

SERIAL

SERIAL

The SERIAL command is used to redirect the console to a serial terminal.

Syntax

`SERIAL [ON [com-port] [baud rate] | OFF]`

com-port

This is a number from 1 to 4 that corresponds to COM1, COM2, COM3, or COM4. The default is COM1.

baud-rate

This is the baud rate to use for serial communications. The default is to have SoftICE automatically determine the fastest possible baud rate that can be used.

Comments

Debugging on a serial console requires a second IBM-compatible PC running MS-DOS. Any PC will do, including 8088, 8086, or 28086 machines. You must first attach the computer to your host machine with a null modem cable. Before using the SERIAL command, you must run the REMOTE.EXE program on the second PC. The syntax of REMOTE.EXE XE “REMOTE.EXE” is as follows:

```
REMOTE com-port baud-rate
```

REMOTE.EXE has two optional parameters. The first parameter is used to specify which com-port on the second PC to use. The allowable com-ports are the same as those listed above. The second parameter is used to specify a baud-rate. For example, to use COM2 at 19200 baud, you would specify:

```
REMOTE 2 19200
```

If no com-port is specified, REMOTE.EXE will use COM1. If no baud-rate is specified, SoftICE will attempt to determine the fastest possible baud-rate that can be used. If a baud-rate is specified for REMOTE.EXE, the same baud-rate must also be specified in the SERIAL command.

Specialized Debugging Commands

SHOW	Display instructions from history buffer
TRACE	Enter trace simulation mode
XT	Single step in trace simulation mode
XP	Program step in trace simulation mode
XG	Go to address in trace simulation mode
XRSET	Reset back trace buffer
VECS	Save/restore/compare interrupt vectors
SNAP	Take snap shot of memory block
EMMMAP	Display EMM allocation map
STACK	Display call stack

SHOW

SHOW

Display instructions from history buffer

Syntax

```
SHOW [B | start]  [L length]
```

B

This tells the show command to start the display with the oldest instruction in the back trace buffer.

start

The number of instructions back from the buffer end (last instruction captured) to begin display.

length

The number of instructions to display

Comments

The SHOW command displays instructions from the back trace history buffer. If source is available for the instructions then the display is in mixed mode, otherwise only code is displayed.

SHOW allows scrolling through the back trace buffer with the up, down, Pageup and PaqeDn keys. To exit from SHOW you must press the Esc key.

Preceding the address of each instruction is the buffer entry number. This number shows how deep into the buffer you are displaying. The higher the number, the deeper you are into the buffer.

Note

Before using the SHOW command, instructions must have been logged with a back trace range. See chapter 9 for more information on back trace ranges.

Hints

It is often useful to have the code window visible with the actual code of the region you are displaying from the back trace buffer. When you compare the actual instruction flow to code, displayed jumps and calls are usually less confusing.

Using SHOW in conjunction with the TRACE command will allow you to see the instructions in the back trace history buffer from two different points of view.

Example

```
SHOW 40
```

This example will displays starting with the 40th instruction back in the back trace buffer.

TRACE

TRACE

Enter trace simulation mode

Syntax

```
TRACE [start] | [OFF]
```

start

The number of instructions back from the buffer end (last instruction captured) to begin trace simulation

OFF

Exit trace simulation mode.

Comments

The TRACE command allows you to replay instructions from the instruction back trace history buffer just as if they were being executed for the first time. To use trace simulation mode you must have the code window visible After entering trace simulation mode you use the XT, XP and XG commands to trace through the instructions in the buffer.

To exit trace simulation mode type TRACE OFF.

TRACE with no parameters specified displays whether trace simulation mode is on or off.

Note

Before using the TRACE command, instructions must have been logged with a back trace range. See chapter 9 for more information on back trace ranges.

Hints

Trace simulation mode is most useful when the code window is visible. It is often useful to use TRACE in conjunction with the SHOW command. This allows the instructions in the back trace history buffer to be viewed simultaneously in two different forms.

Example

```
TRACE 40
```

This example enters trace simulation mode starting 40 instructions back from the last instruction logged. It will remain in trace simulation mode until TRACE OFF is entered.

XT

XT

Single step in trace simulation mode

Syntax

T [R]

R

Single step in reverse direction.

Comments

The XT command single steps through the instruction back trace history buffer. This command acts like the T command for normal debugging. Note that the registers do NOT change while stepping in trace simulation mode except CS and IP,

The XT instruction allows you to replay instructions from the back trace history buffer,

Note

Before using XT you must be in trace simulation mode. See chapter 9 and the TRACE command in this section for more information on back trace ranges.

Hint

If you are using XT frequently, like any other SoftICE command it can be assigned to a function key.

Example

XT

This command single steps one instruction in trace simulation mode.

XP

XP

Program step in trace simulation mode

Syntax

XP

Comments

The XP command does a logical program step through the instruction back trace history buffer. This command acts like the P command for normal debugging. Note that the registers do NOT change while stepping in trace simulation mode except CS and IP.

The XP instruction allows you to replay instructions from the back trace history buffer.

Note

Before using XP you must be in trace simulation mode. See chapter 9 and the TRACE command in this section for more information on back trace ranges.

Hint

If you are using XP frequently, like any other SoftICE command it can be assigned to a function key.

Example

XP

This command executes one program step in trace simulation mode.

XG

XG

Go to an address in trace simulation mode

Syntax

```
X [R] address
```

R

Search for address in reverse direction.

address

Address to go to in the back trace history buffer.

Comments

The XG command moves the instruction pointer to the next occurrence of the specified address in the back trace history buffer. If R is specified preceding the address, then the instruction pointer is moved to the previous occurrence the specified address in the back trace buffer.

The address must be the first byte of an instruction opcode.

The XG is analogous to the G command in normal debugging.

Note

Before using XG you must be in trace simulation mode. See chapter 9 and the TRACE command in this section for more information on back trace ranges.

Example

```
XG 273:1030
```

This command moves the instruction pointer to the next instance of the instruction at address 273:1030.

XRSET

XRSET

The XRSET command resets the back trace history buffer. This command should be executed before setting a back trace range if there is unwanted instruction information in the back trace buffer.

Syntax

`XRSET [A|I|R]`

A

Logs addresses only.

I

Logs addresses and opcodes.

R

Logs addresses, opcodes, and registers.

Comments

The XRSET command resets the back trace history buffer. This command should be executed before setting a back trace range if there is unwanted instruction information in the back trace buffer.

When using the I and R parameters, significantly more memory will be needed for the back trace buffer to store the information. This memory is allocated on the SoftICE device line in CONFIG.SYS.

A program (BTLOG.EXE “BTLOG.EXE”) was added to write the back trace buffer to a file in ASCII text. The amount of information written depends on the option chosen in the XRSET command used to set up the back trace. The syntax of this program is similar to the SHOW command:

```
BTLOG file-name start-line L length
```

Example

`XRSET`

VECS

VECS

Save/restore/compare interrupt vectors

Syntax

VECS [C|S|R]

C

Compare current table with stored table

S

Save current interrupt table to buffer

R

Restore interrupt table from buffer

Comments

The VECS command allows you to save and restore the interrupt table to an internal SoftICE buffer. The actual table can also be compared to the stored table with the differences displayed.

When the C option is used to compare the current interrupt vector table with the stored copy the output is in the following format:

```
address old-vector new-vector
```

Each vector that has changed is displayed.

The interrupt vector table is initially stored when SoftICE is loaded. It is also automatically stored when a program loaded with LDR.EXE. Only one copy of the interrupt vector table is stored, so each time VECS S is executed, previous copy of the interrupt table is overwritten.

If no parameters are specified, the entire interrupt vector table is displayed.

Example

VECS C

This command compares the actual interrupt vector table with one that had been previously stored in the SoftICE internal VECS buffer.

SNAP

SNAP

Take snap shot of memory block

Syntax

`SNAP [C | S | R] address1 address2`

C

Compare buffer with address range

S

Save address range to buffer

R

Restore buffer to address range

Comments

The SNAP command takes a snap shot of a memory block for later comparison. The S option copies a block of memory to a buffer in extended memory. The C option displays differences between the buffer in extended memory and the actual memory specified by the address range. The R option copies the buffer in extended memory to the address range in conventional memory.

When the C option is used to compare the buffer with the address range the output is in the following format:

```
address old-data new-data
```

Each byte that has changed is displayed.

The address is usually not necessary for the C and R options. If the address is not specified, the address from the last time SNAP was entered with a specified address used.

Notes

To use the SNAP command you must have specified the /TRA XXXX switch on the S-ICE.EXE line in CONFIG.SYS.

The SNAP command saves data in the back trace history buffer. If you are using back trace then you will have a conflict with SNAP. Specifically, SNAP will overwrite back trace information if you do a SNAP S when instruction history is in the back trace buffer. Conversely, if you have saved a region with SNAP, then enabling a back trace range will overwrite the SNAP buffer.

Example

```
SNAP S 2000:0 4000:0
```

This command stores the data block from 2000:0 to 4000:0 in the SoftICE back trace buffer.

EMMMAP

EMMMAP

Display EMM allocation map

Syntax

EMMMAP

Comments

The EMMMAP command displays each physical page that is available for EMM memory and the pages that are currently mapped in.

Note

The SoftICE EMM feature must be enabled to use this function. See chapter 8 for more information on enabling EMM capability.

Example

EMMMAP

This example displays the current EMM allocation in in the following form.

Phy page	Seg address	Handle/Page
00	D000	FFFF 01 D400 0001/0000
02	D800	0001/0001
03	DC00	0001/0002

In this example, physical page 0 is located at D000 and is unmapped. Physical page 1 is located at D400 and has handle 1, page 0 mapped into it. Physical page 2 is located at D800 and has handle 1, page I mapped into it. Physical page 3 is located at DC00 and has handle page 2 mapped into it.

STACK

STACK

Display call stack

Syntax

STACK

Comments

A call stack is a list of routines that were called to reach the current address. Using the call stack is especially useful when SoftICE pops up in a library routine. By using the call stack, you can quickly see the last routine in your program that had control before entering the library, even if the program is several levels deep into library calls. The most recently called entry in the stack is displayed first in the command window.

The format of the call stack is:

```
procedure(offset) [line-number]
```

If line-number is a '?' then no line number information was available for this procedure.

The STACK command can only be used if symbolic information is loaded. If the module of an entry in the call stack was not compiled with debug information, no symbolic label will be displayed. Only a hexadecimal offset will be shown.

Windowing Commands

WR	Toggle register window
WC	Toggle/set size of code window
WD	Toggle/set size of data window
EC	Enter/exit code window
.	Locate current instruction
STKWIN	Toggle stack window

WR

WR

Toggle register window

Syntax

```
WR
```

Comments

The command makes the register window visible if not currently visible. If the register window is currently visible, WR removes the register window.

The register window displays the 8086 register set and the processor flags.

Default Function Key:

F2

WC

WC

Toggle/set size of code window

Syntax

```
WC [window-size]
```

window-size

a decimal number between one and 21.

Comments

If window-size is not specified, this command toggles the code window. If it was not visible it is made visible, and if it was visible it is removed.

If window-size is specified the code window is resized, or it was not visible it is made visible with the specified size.

Note

If you wish to move the cursor to the code window use the EC command. See description of the EC command for more details.

Example

```
WC 12
```

If no code window is present, then a code window 12 lines in length is created. If the code window is currently on the screen, it is resized to 12 lines.

WD

WD

Toggle/set size of data window

Syntax

```
WD [window-size]
```

window-size

a decimal number between one and 21.

Comments

If window-size is not specified, this command toggles the data window. If it was not visible it is made visible, and if it was visible it is removed.

If window-size is specified the data window is resized, or it was not visible it is made visible with the specified size.

Example

```
WD 1
```

If no data window is present then a data window of one line is created. If the data window is currently on the screen, it is resized to one line.

EC

EC

Enter/exit code window

Syntax

EC

Comments

The EC command toggles the cursor location between the code window and the command window. If the cursor was in the command window it is moved to the code window, and if the cursor was in the code window it is moved to the command window.

When the cursor is in the code window several options become available that make debugging much easier. The options are

- Point-and-shoot break points

  Point-and-shoot break points are set with the BP command. If no parameters are specified with the BPX command an execution break point is set at the location of the cursor position in the code window. The cursor must be on a line that contains code (place the code window in mixed mode if you are unsure). The default function key assignment for BPX is F9.

- Go to cursor line

  You can set a temporary break point at the cursor and go with the HERE command. The cursor must be on a line that contains code (place the code window in mixed mode if you are unsure). The default function key assignment for HERE is F7.

- Scrolling the code window

  The code window can be scrolled only while the cursor is in the code window. The scrolling keys (UP arrow, DOWN arrow, PageUp and PageDown) are redefined while the cursor is in code window. When the cursor is in the code window the scrolling keys do the following: up -- Scroll code window up one line down -- Scroll code window down one pageup -- Scroll code window up one window pageDn -- Scroll code window down one window

Note

The code window must be visible for the EC command to work.

Default Function Key

F6

.

.

Locate current instruction

Syntax

.

Comments

When the code window is visible, the . command makes the current source line or current instruction visible.

STKWIN

STKWIN

The STKWIN command toggles a stack window inside the SoftICE screen. Each line shows a 4-digit (hex) displacement from BP and the 4-digit (hex) value of the word at that address. You can scroll the stack window using the ALT key in combination with the UP and DOWN arrow keys. You can only edit the highlighted line. To change a value in the stack window, use the ES command.

Syntax

```
STKWIN [ON | OFF]
```

Debugger Customization Commands

PAUSE	Pause after each screen
ALTKEY	Set alternate key sequence to invoke SoftICE
FKEY	Show and edit function keys
BASE	Set/display current radix
CTRL-P	Toggle log session to printer
Print-Screen	Print contents of screen
PRN	Set printer output port

PAUSE

PAUSE

Pause after each screen

Syntax

```
PAUSE [ON | OFF]
```

Comments

PAUSE controls screen pause at the end of each page. If PAUSE is ON, you are prompted to press any key before information is scrolled off the window. The prompt is displayed in the status line at the bottom of the window.

If no parameter is specified, the current state of PAUSE is displayed.

The default is PAUSE mode ON.

Example

```
PAUSE ON
```

This command specifies that subsequent window display commands will cause the screen to wait for you to press a key before scrolling new information off the window.

ALTKEY

ALTKEY

Set alternate key sequence to invoke SoftICE

Syntax

`ALTKEY [ALTletter] | [CTRLletter] | [SYSREQ]`

letter

Any letter (A - Z)

Comments

The ALTKEY command allows the key sequence for popping up SoftICE to be changed. The key sequence be changed to CTRL + letter, ALT + letter, or the SysRq key.

Occasionally you may be using a program that conflicts with the CTRL D key sequence that brings up the SoftICE window. One way to circumvent this possible problem is to use the ALTKEY command to change the key sequence. Another way is to add the SHIFT key to the current sequence. SoftICE does not respond to this key sequence and allows it to go through to your program. For example if a resident program you are using is brought up with the CTRL D key sequence, try using the key sequence CTRL SHIFT D to bring up your resident program. On some keyboards, you must press ALT and the prtsc key simultaneously to generate a system request. Care must be taken so the screen is not printed accidentally.

If no parameter is specified, the current key sequence state is displayed.

The default key sequence is CTRL D.

Example

`ALTKEY ALT Z`

This command specifies that the key sequence ALT Z will now be used to pop up the SoftICE window.

FKEY

FKEY

Show and edit function keys

Syntax

```
FKEY [function-key-name string]
```

function-key-name

F1, F2... F12

string

The string consists of any valid SoftICE commands and the special character ^ (caret) and ; (semicolon). A ^ is placed in the string to make a command invisible. A ; is placed in the string to denote a carriage return.

Comments

The FKEY command is used from the command line to assign a function key to a command string. Function key can be assigned to any command string that can be typed into SoftICE.

If no parameters are specified, then the current function key assignments are displayed.

To unassign a specified function key, use the FKEY command with these parameters: a function-key-name followed by a null string.

The function keys can also be pre-initialized in the definition file S- ICE.DAT. For more information on function key definitions in the definition file, refer to section 6.4.

Using carriage return symbols in a function key assignment string allows you to assign a function key a series of commands. A carriage return is represented by a ; (semicolon).

If you put ^ (shift 6) in front of a function key definition, the subsequent command will be invisible. The command will function as normal, but all information displayed in the command window (including error messages) is suppressed. The invisible mode is useful when a command changes information in a window (code, register or data) but you do not want to clutter the command window,

When a function key is made invisible with ^, the function key can be used in the middle of typing in other command without affecting their operation. For example, if you are using the default assignment for F2, you can toggle the register window with F2 even if you are partially through typing in your next command.

Note

SoftICE now has a definition file named S-ICE.DAT. You can place function key assignments in this file so that function keys will be automatically assigned when SoftICE is loaded. The syntax for assigning a function key in the configuration file is:

```
function-key-name = “string”
```

When assigning function keys to a command string in S-ICE.DAT, the string must be enclosed in double quotes.

Command line examples

FKEY F2 ^WR;

This example will assign the toggle register window command to the F2 key. The ^ makes the function invisible, and the ; ends the function with a carriage return. The F2 key will toggle the register window on or off, and can even be evoked while typing in another command.

FKEY F1 “G CS:120; R; G CS:”

This example shows that multiple commands can be assigned to a single function key and that partial commands can be assigned for the user to complete. After this command is entered, pressing the F1 key will cause the program to execute until location CS:120 is reached, display the registers, then start the G command for the user to complete.

FKEY F1 WD 3;D DS:100;

This example will assign a series of commands to the F1 key. The function is visible, and ends with a carriage return. The F1 key will make the data window three lines long and dump data starting at DS:100.

S-ICE.DAT example

F1 = “WR;WD 2;WC 10;”

If this line is placed in S-ICE.DAT, when SoftICE is loaded it will assign the string to the F1 key. When F1 is pressed while in SoftICE, it will toggle the register window, create a data window of length 2 and a code window of length 10. For more information about assigning function key definitions in S-ICE.DAT, refer to chapter 6.

BASE

BASE

Set/display current radix

Syntax

```
BASE [10 | 16]
```

Comments

The BASE command sets the current radix to base 10 or base 16. Base 10 is of limited use in the narrow window because of window width limitations. It also limits the amount of information displayed in some commands in the wide mode.

When the current radix is base 10, all numbers and addresses typed into and displayed by SoftICE are in decimal, When the current radix is base 16, all numbers and addresses typed into SoftICE are in hexadecimal except

- source line numbers
- screen coordinates and sizes in the WIN command

These exceptions are always typed in and displayed as decimal numbers.

The default radix is base 16.

Example

```
BASE 16
```

This example sets the current radix to base 16.

CTRL-P

CTRL-P

Toggle log session to printer

Syntax

CTRL-P

Comments

When the CTRL key followed by the P key is pressed, all subsequent information displayed in the command window is also sent to the printer. To turn the log to printer mode off, type CTRL followed by P again.

When you are sending a lot of information to the printer using CTRL-P, you may want to turn the PAUSE command OFF to allow information to scroll off the window without pressing a key.

Print-Screen

Print-Screen

Print contents of screen

Syntax

```
Print-Screen
```

Comments

Depressing the print-screen key does a screen dump to printer. All information from the screen is sent the printer.

If you wish to print the memory map or help information is usually much faster to use CTRL-P than Print-Screen. This is because Print-Screen prints every character on the screen including borders.

PRN

PRN

Set printer output port

Syntax

PRN [LPTx | COMx]

x

a decimal number between 1 and 4.

Comments

The PRN command allows you to send output from the CTRL-P and Print-Screen commands to a different printer port.

If no parameters are supplied, PRN displays the currently assigned printer port.

Example

PRN COM 1

This command causes the CTRL-P and Print-Screen command output to go to the COM 1 port.

Screen Control Commands

COLORS	Set display colors
FLASH	Restore screen during P and T
FLICK	Screen flicker reduction
WATCHV	Set watch video mode
RS	Restore program screen
CLS	Clear window
ALTSCR	Change to alternate screen
WIN	Change size of SoftICE window
LINES	Change the size of the display
SL	Toggle display of separator lines
TABS	Set tab expansion of source files

COLORS

COLORS

Set display colors

Syntax

```
COLORS [color | * | r]
```

color

A hex number representing a color to change to

*****

Skip changing the color for this field

r

Restore the original colors specified in S-ICE.DAT

Comments

The COLOR command accepts up to 12 parameters in the following order:

normal - highlight - reversefor Register window

normal - highlight - reverse for Data window

normal - highlight - reverse for Code window

normal - highlight - reverse for Command window

The colors will only stay the way you change them until the computer is rebooted. At that time, the colors will revert to the default colors. To change the colors on a permanent basis, see the color line in S-ICE.DAT.

Example

```
COLORS***4f 40 74***5d
```

This would leave the Register and Code window attributes the same, change the Data window attributes to 4F for normal, 40 for highlight, and 74 for reverse, and change the Command window normal attribute to 5D, leaving the highlight and reverse.

FLASH

FLASH

Restore screen during P and T

Syntax

```
FLASH [ON | OFF]
```

Comments

The FLASH command lets you specify whether the screen will be restored during any Trace and Program step commands. If you specify that the screen is to be restored it is restored for the brief time period that the P or T command is executing. This feature is needed to debug sections of code that access video memory.

If the P command executes across a call or an interrupt, the screen will always be restored, because the routine being called may write to the screen.

If no parameter is specified, the current state of FLASH is displayed.

The default is FLASH mode OFF.

Example

```
FLASH ON
```

This command turns on FLASH mode. The screen will be restored during any subsequent P or T commands.

FLICK

FLICK

Screen flicker reduction

Syntax

```
FLICK [ON | OFF]
```

Comments

Certain types of video cards require waiting for horizontal or vertical retrace before outputting characters. If the video writes are made arbitrarily, flickering will appear while displaying characters. If flickering occurs on your screen while using the SoftICE window, you should turn FLICK on.

With some EGA cards, colors will not be restored properly when you exit from SoftICE. This is a problem with virtualizing EGA video. The port 3DA is a video port used for two purposes. The first is old CGA software polling 3DA for hsync and vsync. This allows them to have flicker free output on some old CGA controller cards. The second is that it is used to reset a palette latch on EGA cards. SoftICE has an algorithm to avoid having to constantly watch this port, which would slow down old programs that think they are on a CGA. However, there can occasional be circumstances where this algorithm does not work. If you are using Soft- ICE on an EGA screen and you notice that the colors are not restored correctly, then turn FLICK ON and SoftICE will watch the 3DA port, fixing the problem.

When FLICK mode is ON, screen update will be slower.

If no parameter is specified, the current state of FLICK is displayed.

The default is FLICK mode OFF.

Example

FLICK ON This command turns on FLICK mode. This causes SoftICE to wait for the horizontal or vertical retrace before outputting characters.

WATCHV

WATCHV

Set watch video mode

Syntax

```
WATCHV [ON | OFF]
```

Comments

The WATCHV command allows you to specify how SoftICE should watch the video ports. Normally, SoftICE only watches video ports after an INT 10 instruction has been executed that switches to a non-character video mode. Some programs do not use INT 10 to switch modes. In these cases, if WATCHV is OFF, SoftICE may have trouble saving and restoring the screen properly. Turning WATCHV ON will cause SoftICE to watch the video ports all the time.

Turn WATCHV ON if you notice that SoftICE is not handling your screen properly, or if the cursor is not being restored properly. Turning WATCHV ON may have a performance impact in certain video modes.

If no parameter is specified, the current state of WATCHV is displayed.

The default is WATCHV mode OFF.

Example

```
WATCHV ON
```

This command turns on WATCHV mode. This causes SoftICE to watch additional video ports for the purpose of virtualization.

RS

RS

Restore program screen

Syntax

RS

Comments

The RS command allows you to restore the program screen temporarily. The SoftICE window disappears until any key is pressed.

This feature is useful when debugging graphic programs that update the screen frequently. When SoftICE is brought up, it returns to text mode. Using the RS command temporarily restores the graphics screen.

Example

RS

CLS

CLS

Clear window

Syntax

CLS

Comments

The CLS command clears the SoftICE window and moves the prompt and the cursor to the upper left-hand corner the window.

Example

CLS

ALTSCR

ALTSCR

Change to alternate screen

Syntax

```
ALTSCR [ON | OFF]
```

Comments

The ALTSCR command allows you to redirect the SoftICE output from your default screen to the alternate screen. This feature is useful, for instance, when you want to debug a graphics program without having to switch between the SoftICE window and the graphics display.

ALTSCR requires the system to have two monitors attached. The alternate monitor should be in a character mode, which is the default mode for monitors.

The default is ALTSCR mode OFF.

Example

```
ALTSCR ON
```

This command redirects screen output to the alternate monitor.

WIN

WIN

Change size of SoftICE window

Syntax

```
WIN [N | W] [start-row length [start-column]]
```

N

When N is specified, the window will be set to the narrow width: 46 characters.

W

When W is specified, the window will be set to full screen width.

start-row

Number from 0 to 17 specifying row where window display starts.

length

Number from 8 to 25 specifying how many lines tall you want the window to be.

start-column

Column position of the left side of narrow window. The start-row and start-column specify the upper left hand corner of the narrow window. The start-column is ignored if applied to the wide window.

Comments

The WIN command allows you to modify the width and height of the SoftICE display window.

If no parameters are specified, this command toggles the window between wide and narrow screen display modes.

If the WIN command is specified with only the N or the W parameter, the window size will be changed to the requested width at the current height.

If the number of lines plus the starting row number is larger than 25, the window length goes to the bottom of the screen.

The default is WIN mode narrow.

Examples

```
WIN N 4 9 30
```

This command causes the window display to start at row 4 and column 30, and to be 9 rows tall and 46 characters wide.

WIN

This command toggles the window display width from its current state (either wide or narrow) to the opposite state.

WIN W 10 8

This command causes the window display to start at row 10, and to be 8 rows tall and go the width of the screen.

LINES

LINES

The LINES command changes the number of lines of SoftICE's character display. It allows three different display modes: 25-line, 43-line, or 50-line.

Syntax

```
LINES [25 | 43 | 50]
```

Comments

The default number of display lines is 25. If your SoftICE display is on another computer connected by a serial cable, the display is fixed at 25 lines. 43-line mode is only valid on VGA or EGA display adapters. 50-line mode is only valid on VGA adapters.

SL

SL

The SL command puts separator lines between each of the information windows in SoftICE for DOS. This is very useful when debugging on a monochrome monitor where there are no colors to separate the windows for you. SL is a toggle for the lines. Type SL once and the lines are displayed, type it again and the lines are removed.

TABS

TABS

Control tab expansion size of source files.

Syntax

```
TABS [2 | 4 | 8]
```

Comments

If no parameter is specified then the current tabs setting is displayed. An interesting use of the TABS command is to see more than 78 characters of source on a single line. To do this enter TABS 2.

Symbol and Source Line Commands

SYM	Display/set symbol
SYMLOC	Relocate symbol base
SRC	Toggle between source, mixed and code
FILE	Change/display current source
SS	Search current source file for string

SYM

SYM

Display/set symbol

Syntax

```
SYM [symbol-name [value]]
```

symbol-name

A valid symbol name. The symbol name can end with an * (asterisk). This allows searching if only the first part of the symbol name is known. The , (comma) character can be used as a wild card character in place of character in the symbol-name.

value

This is a word value that is used if you want to set a symbol to a specific value.

Comments

The SYM command allows displaying and setting of symbols. If SYM is entered with no parameters all symbols are displayed. The value of each symbol is displayed next to the symbol name.

If a symbol name is specified with no value then the symbol name and value are displayed. If the symbol name was not found then nothing is displayed.

The SYM command is often useful for finding a symbol name when you can only remember a portion of the name Two wild card methods are available for locating symbols. If symbol-name ends with an *, then all symbols that match the actual characters typed prior to the * will be displayed regardless of their ending characters. If a , is used in place of a specific character in symbol-name, that character is a wild card character.

If value is specified, all symbols that match symbol-name are set to the value. All symbols have word values.

Examples

```
SYM FOO*
```

All symbols that start with FOO are displayed.

```
SYM FOO* 6000
```

All symbols that start with FOO are given the value 6000.

SYMLOC

SYMLOC

Relocate symbol base

Syntax

```
SYMLOC segment-address
```

Comments

The SYMLOC command relocates the segment components of all symbols relative to the specified segment address. This function is necessary when debugging loadable device drivers or other programs that can not be loaded directly with LDR.EXE.

When relocating for a loadable device driver, use the value of the base address of the driver as found in the MAP command. When relocating for an .EXE program, the value is 10H greater than that found as the base in the MAP command. When relocating for a .COM program, use the base segment address that is found in the MAP command.

The MAP command will display at least two entries for each program. The first is typically the environment and the second is typically the program. The base address of the program is the relocation value.

Example

```
SYMLOC 1244 + 10
```

This will relocate all segments in the symbol table relative to 1244. The + 10 is used to relocate a TSR that was originally a .EXE file. If it is a .COM file the + 10 is not necessary.

SRC

SRC

Toggle between source, mixed and code

Syntax

SRC [?]

Comments

The SRC command toggles between source mode, mixed mode and code mode in the code window.

If SRC ? is entered, the current state is displayed.

Example

SRC

This command changes the current mode of the code window. If the mode was source, it becomes mixed. the mode was mixed, it becomes code. If the mode was code, it becomes source.

Default-Function Key

F3

FILE

FILE

Change/display current source file

Syntax

```
FILE [file-name |*]
```

Comments

If a file-name is specified, that file becomes the current file and the start of the file is displayed in the code window. If no name is specified, the name of the current source file (if any) is displayed.

The FILE command is often useful when setting a break point on a line that has no associated public symbol. Use file to bring the desired file into the code window, use the SS command to locate the specific line, move the cursor the specific line, then type BPX to set the break point.

FILE * displays all source files that have been loaded by LDR.EXE into extended memory.

Note

Only source files that have been loaded into extended memory with LDR.EXE are available with the FILE command.

Example

```
FILE MAIN.C
```

If MAIN.C had been loaded with LDR.EXE, this command brings it up in the code window starting with line 1.

SS

SS

Search current source file for string

Syntax

```
SS [line-number] [‘string’
```

line-number

a decimal number

string

a character string surrounded by quotes (single or double)

Comments

The SS command searches the current source file for the specified character string.

The search starts at the specified line number. If no line number is specified, the search starts at the first line displayed in the code window.

If no paramaters are specified, the search continues for the previously specified string.

Note

The code window must be visible and in source mode.

Example

```
SS 1 ‘if (i==3)’
```

5 SoftICE Initialization Options

Introduction

The SoftICE program file (S-ICE.EXE) can be loaded as a loadable device driver in CONFIG.SYS or as a program from the DOS command line. To get the full power of SoftICE, it must be initially loaded as a device driver in CONFIG.SYS. However, there may be circumstances when you might want to run SoftICE from the DOS prompt or a batch file, such as:

- You do not have extended memory in your system SoftICE can only load as a loadable device driver if you have extended memory.
- You want to take up ZERO bytes of conventional memory. When loaded as a device driver, SoftICE occupies approximately 2K of conventional memory.
- You only need to use SoftICE occasionally and there are no other programs using extended memory.

In some cases you may need some of the features that require SoftICE to be loaded in CONFIG.SYS but do not want SoftICE to be resident all of the time. In this case SoftICE can be loaded in CONFIG.SYS to reserve extended memory, and then disabled, by using the /UN switch, until Soft- ICE is required. See *SoftICE Loading Switches* on page 143 for more information about the /UN switch.

Loading SoftICE from the DOS Prompt

You can NOT enable all of SoftICE's features when loading from the DOS prompt. If you will be using SoftICE as a stand-alone debugger, it is recommended you load SoftICE in the CONFIG.SYS file.

To load SoftICE from the DOS prompt type:

◊ S-ICE

In systems with no extended memory present, SoftICE loads itself at the highest memory location possible. The memory used by SoftICE is then 'mapped out', making it invisible to DOS programs. Since the total memory visible to DOS and its programs is less after SoftICE loads, it is recommended that you load SoftICE before any TSR's control programs.

In systems with extended memory, you should only load SoftICE from the DOS prompt if you are not using extended memory for anything else (e.g., VDISK, CACHE, HIMEM...). When you initially load SoftICE from the command line or from a batch file, SoftICE will prompt you with a warning message. This warning message is just to remind you that SoftICE will overwrite the highest portion of extended memory when it loads. You can suppress this warning prompt with the EXTENDED option in the SoftICE configuration file S-ICE.DAT. For more information about the EXTENDED option, see *Special Configuration Options* on page 145.

Loading SoftICE as a Loadable Device Driver

In order to use all of the SoftICE features, you must first load SoftICE as a loadable device driver in your CONFIG.SYS file. The features this makes possible are:

- Coexisting with other software that uses extended memory.

  Loading as a device driver allows SoftICE to manage extended memory so you can run SoftICE with programs that use extended memory, such VDISK, CACHE and HIMEM.

- Symbolic and source level debugging

  Loading as a device driver allows SoftICE to allocate an extended memory buffer for symbols and source information.

- Back trace ranges and the SNAP command

  Loading as a device driver allows SoftICE to allocate an extended memory buffer for a back trace buffer. This buffer is also used for the SoftICE SNAP command.

- Enabling SoftICE's EMM 4.0 capability

- Running SoftICE with MagicCV or MagicCVW

Note: When loaded as a device driver in CONFIG.SYS, SoftICE allocates the highest portion of extended memory for itself and its associated components, so there can be no memory conflicts. S-ICE.EXE must be loaded in CONFIG.SYS before any other driver that allocates extended memory loaded (e.g., VDISK.SYS, RAMDRIVE.SYS). Generally SoftICE works best if it is the first loadable device driver installed in CONFIG.SYS.

SoftICE Loading Switches

One or more loading switches can follow S-ICE.EXE in CONFIG.SYS. These switches allow you to customize the way extended memory will be reserved by SoftICE. The switches all must begin with a / character. The loading switches are:

- /EXT XXXX -- Informs S-ICE.EXE to reserve XXXX Kilobytes of extended memory for other DOS programs that use extended memory (e.g., VDISK, CACHE, HIMEM,...). If the /EXT switch is not present, then any extended memory not used by SoftICE and its associated components will be left as standard extended memory, but the amount can not be guaranteed. The /EXT switch is useful because it is sometimes difficult to determine exactly how much memory being used by SoftICE and its associated components. Using the /EXT switch will guarantee a specified amount is available for other programs that use extended memory.
- /SYM XXXX -- Informs S-ICE.EXE to reserve XXXX Kilobytes of extended memory for symbols and source usage. If XXXX is not specified, then all remaining extended memory is used for symbols. Enough memory must be allocated for your .SYM file and all source files. For more information about using symbols and source, see *Chapter 6: Symbolic and Source Level Debugging* on page 149.
- /TRA XXXX -- Informs S-ICE.EXE to reserve XXXX Kilobytes of extended memory for a back trace history buffer. This buffer is used for back trace ranges and for the SNAP command. If XXXX is not specified, then 10K of extended memory is automatically reserved for the buffer. If you do not want any memory reserved for a back trace buffer, use /TRA 0. For more information about using back trace ranges, see *Chapter 8: Back Trace Ranges* on page 165.
- /MCV XXX -- Informs S-ICE.EXE to reserve XXX Kilobytes of extended memory for MagicCV or MagicCVW. The minimum amount of extended memory you can specify is 280K and the maximum is 620K. If XXX is not specified, S-ICE.EXE will reserve the remaining memory, between 280K and 620K. See *Chapter 9: Using SoftICE with MagicCV or MagicCVW* on page 169 for more information about running SoftICE with MagicCV or MagicCVW.
- /EMM XXXX -- Informs S-ICE.EXE to turn XXXX Kilobytes of extended memory into EMM 4.0 conforming expanded memory. If XXXX is specified, then all remaining memory is used as expanded. See *Chapter 7: Expanded Memory Support* on page 157 for more information about expanded memory support.

- /UN -- Informs S-ICE.EXE to enter protected mode, reserve any needed extended memory, then exit protected mode and unload itself. This switch should be used when you are loading S-ICE.EXE as a loadable device driver, but you don't want your system to remain in protected mode. This switch will reserve memory for SoftICE, and you must execute S-ICE.EXE from the DOS prompt when you are ready to use SoftICE.

SoftICE reserves extended memory in the following order, regardless of the order the switches are specified:

- ◊ Reserve approximately 120K for S-ICE.EXE.
- ◊ Reserve memory for the /EXT switch if present.
- ◊ Reserve memory for the /SYM switch if present.
- ◊ Reserve memory for the /TRA switch if present. if it is not present, default to reserve 10K for the back trace buffer.
- ◊ Reserve memory for the /MCV switch if present. Reserve memory for the /EMM switch if present.

If available memory runs out while trying to reserve memory for a switch in the above sequence, then S-ICE.EXE does the following:

1. The remaining extended memory is allocated to switch being processed when memory runs out.
2. No memory will be reserved for the remaining switches.

Note: If the /MCV or /EMM switch is present, a additional 64K of extended memory is reserved for a DMA holding buffer.

The switches can be placed in any order following DEVICE = S-ICE.EXE. example is:

```
DEVICE = S-ICE.EXE /TRA50 /EMM 500 /SYM 2048
```

If four megabytes of extended memory are available, this example will reserve approximately 120K for SoftICE, 2 megabytes for symbols, 50K for a back trace history buffer, 500K for expanded memory and leave approximately 1.3 megabytes for other extended memory programs. Note that SoftICE will load into the highest portion of extended memory, leaving the remaining memory starting at 100000H (one megabyte mark).

The SoftICE Initialization File S-ICE.DAT

SoftICE has several load options. These options are specified by placing special commands in an initialization file named S-ICE.DAT. S-ICE.DAT is an ASCII text file that SoftICE parses at load time. This file can contain function key assignment an auto-start string and various configuration options. The file can be created and edited with any DOS text editor. When

loading SoftICE from the command line, S-ICE.DAT must be placed in the current directory or in a directory that is accessible through your current PATH. When SoftICE is loaded as a device driver in CONFIG.SYS, S-ICE.DAT must be in the same directory where S-ICE.EXE is located.

There are three categories of commands that can be included in the S- ICE.DAT initialization file:

- Special configuration options
- Function key assignments
- Initialization command sequence

Special Configuration Options

Any of the following configuration options that are needed should each be placed on a separate line in the S-ICE.DAT file.

- COMPAQ -- Compaq 386 and 386SX computer and some Compaq compatible computers (including computers containing Micronix motherboards) have 384K of non-contiguous extended memory. The COMPAQ option is necessary if you want SoftICE to use this memory. Note that the COMPAQ option is the same as the /C command line parameter in SoftICE 1.X.
- NOLEDS -- The NOLEDS option tells SoftICE not to set and clear the keyboard LEDs while the SoftICE window is up. On some keyboards the are timing problems that will cause SoftICE to lose synchronization with the keyboard. If SoftICE hangs when you are in the SoftICE window use this option. Note that the NOLEDS option is the same as the /L command line parameter in SoftICE 1.X.
- NOTVGA -- The NOTVGA option allows SoftICE to run on BIOS compatible VGA cards. Many VGA cards are not compatible with IBM VGA at the hardware level. These cards support VGA at the BIOS level only. Use this switch if you have one of those video adapters. Note that the NOTVGA option is the same as the /V command line parameter in SoftICE 1.X.
- EXTENDED -- The EXTENDED option causes SoftICE to load directly into extended memory without prompting the user with a warning message. It should be used if you are loading SoftICE initially from the DOS prompt and do want to be prompted, and you know nothing else using extended memory. Note that the EXTENDED option is the same as the /E command line parameter in SoftICE 1.X.

Function Key Assignments

One or more SoftICE commands can be assigned to any function key at load time. See the description of the FKEY command in Debugger Customization Commands for a description of assigning function keys from the SoftICE command line.

The syntax for assigning a function key name in S-ICE.DAT is :

function-key-name = “string”

function-key-name -- F1, F2... F12.

string -- The string may consist of any valid SoftICE commands and the special characters ^ and ;. A ^ is placed in the string to make a command invisible. A ; is placed in the string denote a carriage return. The string must be enclosed in double quotes.

An example function key assignment in S-ICE.DAT is:

F12 = “D 100;”

This will assign the SoftICE dump command to function key 12. When F12 is pressed SoftICE will dump at offset 100H in the current data segment. The semi-colon following the 100 represents the ENTER key.

Initialization Command Sequence

A sequence of commands can be automatically executed when SoftICE loads. This is useful for customizing SoftICE to meet your needs. For example, you might set up windows and change the default hot key sequence. The syntax for setting up an initialization command sequence in S-ICE.DAT is:

INIT = “assignment-string”

assignment string -- The string consists of any valid SoftICE commands and the special characters ^ and ;. A ^ is placed in the string to make a command invisible. A; is placed in the string denote a carriage return. The string must be enclosed in double quotes.

An example initialization command sequence in S-ICE.DAT is:

INIT = “WIN; WR; WD 1; WC 12; ALTKEY CTRL X;”

This example will put the SoftICE window in full screen mode, create a register window, create a data window one line long, create a code window 12 lines long, and change the hot key sequence to CTRL X.

Sample S-ICE.DAT

A sample S-ICE.DAT initialization file is included on the distribution diskette. This sample assigns the function keys so they are used in a similar manner as the function keys in Microsoft’s CodeView debugger. This sample S-ICE.DAT should also be used as is for the tutorial in chapter 3.

Using .PTH and .SRC Files

If there is no SET SRC=Ö XE “SET SRC=Ö” statement within the environment, LDR.EXE XE “LDR.EXE” will attempt to open a file named program-name.PTH XE “.PTH”. If such a file exists, LDR will read it and will use the paths specified within to look for the source files. In version 2.8, this was modified to make LDR.EXE look for a program-name.PTH file first, so that it supersedes any SET SRC statement.

Before loading the source files, LDR.EXE will attempt to open a file named program SRC XE “SRC”. This file should be a list of the source files to load – if the first file exists, only those source files listed within it will be loaded.

The .PTH and .SRC files must be in the same directory as the program-name.EXE file. The syntax of a .PTH file is like a path statement except that it has no “PATH=”, but just the paths. The following is an example:

```
c:\ldr\new;d:\bcdos\engine;c:\sibcdos\ui;
```

The format of a .SRC file is a list of source file names, including extensions, each one on a separate line – for example:

```
asm.asm
sym.asm
volume.cpp
mgraph.c
```

Note that this file does not contain any drive or directory information.

Using Borland .TDS Files XW “Borland TDS Files” for Debug Data

If a Borland .EXE file does not contain debug data, LDR.EXE will try to open a file named program-name.TDS in the same directory as the .EXE file. If such a file exists, it will read the debug data from this file.

A TDS file can be generated by running TDSTRIP XE “TDSTRIP” with the /s switch on a .EXE file which contains Borland debug data.

Passing Commands to SoftICE from LDR.EXE

LDR.EXE can pass commands to SoftICE in either of two ways: directly from the command line, or from a file specified on the command line. In both cases, the command string is executed when SoftICE pops up at the start of the application program. If you want the program to execute without stopping at the SoftICE screen, put an “X” as the last command in the string. To pass commands directly from the command lone, the entire command string

must be enclosed within slashes. The command string may include symbolic names and commands to set break points. For example, the following string would set a breakpoint on a variable named “base,” set up the Data window to display this variable in word format, and then exit from SoftICE and begin executing he application program without popping up:

```
ldr /bpm base;dw base;x/ myprog
```

Note that there is a slash both before and after the command string – this allows using spaces within commands. Also note that commands are separated by semicolons. The last command in the string does not need a trailing semicolon.

To specify a file containing SoftICE commands, use “/n” or “/N” switch in the LDR command line (There are no SoftICE commands which begin with “n” or “N,” so this cannot be a direct command.). There must NOT be a slash after the file name – the first space after the file name is the terminator. The command file either must be in the current directory or else the full path must be specified. The command string within the file may use either CR’s or CRLF’s or semicolons as separators between commands. The size of the SoftICE command buffer limits the file length to 200 characters.

The syntax would be something like the following:

```
ldr /nc:\prog\softice.cmd myprog
```

6 Symbolic and Source Level Debugging

Introduction

SoftICE can load programs, symbol tables and source files for enhanced debugging. Symbolic debugging allows you to set break points and reference variables with symbol names rather than specifying numeric addresses. Source level debugging allows you to step through your program at the source code level rather than assembly code level.

Symbol and source line number information is extracted from the link map file. The link map must be compatible with Microsoft's linker version 3.60 or greater.

Symbols and source files reside in extended memory. You must have sufficient extended memory for the symbols and source files. Source files are not paged from the disk as in many debuggers. This allows SoftICE to provide complete system debugging in source level, You can debug T&SR's interrupt routines and other systems level code at the source level.

Note: You cannot use symbolic or source level debugging unless SoftICE has been loaded as a device driver in CONFIG.SYS.

Preparing for Symbolic or Source Debugging

Before debugging a program with symbols or source you must create a symbol file. This is a binary file that contains symbol and line number information in a format that SoftICE can understand. This file is created with the utility MSYM.EXE. MSYM.EXE reads in your link map to create a symbol file with the extension (.SYM).

Preparing for Symbolic Debugging Only

To prepare a program for symbolic debugging only, you must do the following steps:

1. Compile or assemble your program.
2. Link your program with the proper switches to create a .MAP file that contains a list of public symbols. If you are using Microsoft's linker, the /MA switch is the proper switch to use. This .MAP file must be identical to the .MAP file produced by Microsoft's linker, version 3.60 or greater.
3. Create a.SYM file by running MSYM.EXE. The syntax for using MSYM.EXE is:

   ◊ MSYM program-name [.extension]

   If the extension is not supplied MSYM assumes the extension is.MAP. MSYM reads in a map file as in and writes out a symbol file as output. The symbol has the name program-name.SYM.

Note: Before compiling or assembling your program you may want to make some additional symbols public. Only public symbols are supported with SoftICE symbolic debugging. The way to make a variable or a label public varies, depending upon which language you are using.

In 8086 assembly language, simply use the PUBLIC directive followed by the locally defined symbols you wish to make public. For example:

```
PUBLIC FOO, LOOP1, STATUS
```

In C language, all procedure names and static variables are defined outside a block are public.

For other languages, refer to your language manual for details.

Preparing for Symbolic and Source Level Debugging

To prepare a program for both symbolic and source debugging, you must do the following steps:

1. Compile or assemble each module that you wish debug at the source level with the appropriate switch to put line number information into the object files. With Microsoft languages you can use either the /Zi or the /Zd switches. You may not want to do this with all files, because the combined file sizes of the symbol file and all the source files compiled with these switches must fit into the amount of extended memory you have reserved with the /SYM loading switch in CONFIG.SYS.
2. Link your program with the proper switches to create a.MAP file that contains source line numbers and a list of public symbols. If you are using Microsoft's linker, the /LI and /MA switches are the proper switches to use. This .MAP file must be identical to the.MAP file produced by Microsoft's linker, version 3.60 or greater.

3 Create a.SYM file by running MSYM.EXE. The syntax for using MSYM.EXE is:

◊ MSYM program-name [.extension]

If the extension is not supplied MSYM assumes the extension is.MAP. MSYM reads in a map file as input and writes out a symbol file as output. The symbol file has the name program-name.SYM.

Reserving Memory for Symbols and Source Files

Before loading programs, symbol files and source files you must reserve extended memory for them. Extended memory is reserved when you load Soft- ICE in CONFIG.SYS. Before reserving extended memory you may want to add up the file sizes of the .SYM file and all of the source files that you want to load. You must reserve at least this much extended memory. You must use the /SYM loading switch when loading S-ICE.EXE. A sample line in CONFIG.SYS for loading SoftICE and reserving space for symbols and source files is:

DEVICE = S-ICE.EXE /SYM 1024

This example loads SoftICE into extended memory and reserves 1 megabyte of memory for symbols and source files. See section 6.3 (Loading SoftICE as a Loadable Device Driver) for more details on reserving memory.

Loading Programs and Symbol Files

The SoftICE utility LDR.EXE is used for loading programs, symbol files and source files. For symbolically debugging application programs and T&SR programs you will typically use LDR.EXE to load the program, symbols and source files in one step. For debugging loadable device drivers, ROMs and other system components you will typically use LDR.EXE to load the symbol file and source files only.

The syntax for LDR.EXE is:

◊ LDR program-name | program-name.SYM | program-name.extension

Loading Program, Symbols and Source

To load your program, symbols and source files in one step, you must use LDR.EXE in the form:

◊ LDR program-name

Notice that program-name does not have a file extension. If no file extension is supplied, then LDR.EXE will do the following:

1. Load program-name.SYM into extended memory
2. Load source files into extended memory. This step is done only if source records exist in the .SYM file.
3. Load program-name.EXE into memory at the location it would have loaded if it had been loaded directly from the DOS prompt.
4. Bring up SoftICE with the instruction pointer at first instruction of your program. If it is a C program and source is loaded for the file containing , _MAIN, then the source for that file will be visible in the code window.

Loading Only Symbols and Source Files

If you wish to load only symbols and source files (for debugging a loadable device driver for example) you must use LDR.EXE in the form:

◊ LDR program-name.SYM

Notice that the.SYM extension is specified. This will load the .SYM file and source files into extended memory. When symbols are loaded by this method your program or device driver symbols are assumed to be referenced from 0:0. Since this is rarely the case you will need to use the SoftICE command SYMLOC to locate the symbols. See the description of the SYMLOC command in section 5.10 for a complete description. An example of loading a symbol file called DRIVER.SYM is:

◊ LDR DRIVER.SYM

Loading a Program With No Symbols or Source

To load a program file without loading the associated symbol file you must use LDR.EXE in the form:

◊ LDR program-name.extension

Notice that the file extension is present. Typically the file extension will be.EXE or.COM. When a file extension specified LDR.EXE will load the program and bring up SoftICE with the instruction pointer at the first instruction of the program. An example of loading a program with symbols and source is:

◊ LDR TEST.EXE

Notes: LDR.EXE saves a copy of the interrupt vector table automatically when it loads your program. This is equivalent to doing a VECS S command. If you are going to exit your program before it runs to completion, you can do an EXIT R to exit the program and restore the interrupt vector table. Using LDR.EXE to load only the program-name.EXE is often useful for restarting your program while in the middle of a source level debugging session. To restart, the EXIT R command to abort the current session. Then use LDR.EXE to reload your.EXE file. The symbols: source do not have to be loaded since they remain in extended memory.

If LDR.EXE gives you the message “Out of space loading symbol information”, this means that you did not reserve enough extended memory with the /SYM loading switch in CONFIG.SYS.

If LDR.EXE does not find your source files on the same directory as the program you are loading, LDR.EXE will prompt you for the path names where it can find the source files. If you have source files on several directories or are loading a program frequently this becomes cumbersome. You can eliminate the need for prompting by using the DOS environment variable SRC. LDR.EXE uses this environment variable to find source files before prompting the user. The syntax for setting the environment variable from the DOS prompt is:

◊ SET SRC = directory;directory;...;directory

Each of the specified directories will be searched before the user is prompted.

Limitations:

- SoftICE supports symbols for only one program at a time. If you load a new .SYM file, the existing one is overwritten.
- SoftICE does not follow overlays or Microsoft Windows segment movement.
- SoftICE recognizes public symbols and line numbers only. It does not support local variables.

Multiple Symbol Tables

SoftICE can handle two symbol tables. This is useful when debugging a T&SR or DOS loadable device driver with an application, or debugging a shell with a child process.

To load a separate symbol table or a separate program with symbols use the SoftICE TABLE command. TABLE 1 uses symbol table number one, TABLE 2 uses symbol table number 2.

To use two symbol tables, do the following:

1. Use LDR to load your first program and symbolic information.
2. Pop up SoftICE.
3. Enter Table 2.
4. Exit SoftICE.
5. Use LDR.EXE to load the second symbol table.

Both sets of symbolic information are now loaded into extended memory and you are currently viewing the second set of symbolic information. Use the TABLE 1 and TABLE 2 commands to toggle between which set of symbolic information you are currently viewing.

To view your first program's symbolic information, pop up SoftICE if it's not up already, and enter:

```
TABLE 1
```

To view your second program's symbolic information, pop up SoftICE if it's not up already, and enter:

```
TABLE 2
```

If you enter TABLE without any parameter, it will tell you which set of symbolic information is currently being viewed.

Note: When you re-load table 1 by entering TABLE 1 then loading with LDR, TABLE 2 is invalidated.

Debugging With Symbols

After you have loaded your program and.SYM file you can begin debugging your program symbolically. In general a symbol can be used in any command in place of an address.

Symbols are also used by several SoftICE commands when addresses are displayed. For example, the U command displays symbol names of labels and procedures as it encounters them.

There are two commands that are helpful when you are symbolically debugging:

- SYM -- Use the SYM command to get a listing of symbol names and values, or to change the value a symbol. *

- SYMLOC -- Use the SYMLOC command to relocate the base of all of your symbols. You would need to use the SYMLOC command when:
  - ◊ Loading symbols for a loadable device driver
  - ◊ Loading symbols for a T&SR that has already been loaded
  - ◊ Your program moves itself to a location other than its original location. See section 5. 10 for a complete description of these commands.

Debugging With Source

When source files are loaded, SoftICE allows you to view and step through your source code as you are debugging. SoftICE offers two different modes of source level debugging: mixed mode and source mode. Use the SRC command to switch between modes.

Mixed mode shows source lines and the assembly language produced by those source lines intermixed on the display. Mixed mode is useful when you must debug at the assembly level, but use the source lines for reference. Mixed mode is allowed whether the code window visible or not.

Source mode strictly shows source lines on the display. Source level debugging requires the code window to be visible.

Using Line Numbers

Line numbers can be used in place of addresses in several commands. To differentiate a line number from an actual address, place a . (period) in front of the number. For example, to set an execution break point at source line 45 type:

```
BPX .450
```

Using Source Mode in the Code Window

The code window must be visible to enter source mode. If not visible, use the WC command to make it visible. Once you are in source mode you can use SoftICE commands switch to a different source file, view source at any location in the file, scroll through the file, search for strings in the file, and set break points in the file. For a complete description of the following commands see their command descriptions in chapters 4 and 5. The following list is a brief overview of commands that are useful when debugging source code:

- Make the code window visible (if it is not already) with the WC command.
- Toggle between source, mixed, and code modes with the SRC command. To toggle modes enter:
  - ◊ SRC 186

- Place a source file in the code window (if it is n@ already) with the FILE command. For example change from the current file to file MAIN.C enter:
  - ◊ FILE MAIN.C
- Display source at a specific location within the source file with the U command. To change the view to a specific line number or memory address use the U command. You can specify actual addresses or line numbers as a parameter to the command. For example, to view source in the code window starting at source line 450 enter:
  - ◊ U .450
- Locate the current instruction in the code wind@ with the . (period) command.
- Search for a specific character string with the S@ command. For example, to search for the string “Hello World” starting at line 100 in the current source file enter:
  - ◊ SS 100 “Hello World”
- Move the cursor to the code window (if it is not already) with the EC command.
- Scroll the source with the keys up, down, PaqeUp, PageDn.
- Set point-and-shoot break points with the BPX command. Simply place the cursor on the source line that you wish to break on, then enter:
  - ◊ BPX

7 Expanded Memory Support

Introduction

SoftICE has an expanded memory manager built into its kernel. The SoftICE expanded memory manager supports the Lotus-Intel-Microsoft 4.0 specification. This SoftICE feature is useful if you are using programs that support the EMM specification, or if you must backfill your conventional memory to extend your conventional memory to 640K or more.

Other 386 control programs that provide EMM capability (such as QEMM or 386-to-the-MAX) will not co-exist with SoftICE. If you are using those programs for EMM capability or backfilling, you can use the SoftICE EMM manager in their place.

Enabling EMM capability in SoftICE involves the following steps:

1. Configure the expanded memory environment with the utility EMMSETUP.EXE. This utility modifies S-ICE.EXE with the desired EMM page map.

2. Add the /EMM switch to your S-ICE.EXE line CONFIG.SYS. This reserves a portion of extended memory for expanded memory. An example line in CONFIG.SYS that reserves memory for EMM is:

   DEVICE = S-ICE.EXE /EMM 2048

This will reserve 2 megabytes of extended memory for EMM use. See *Loading SoftICE as a Loadable Device Driver* on page 142 for details of installing SoftICE in CONFIG.SYS.

3. Reboot your system.

Configuring The EMM Environment

Before installing S-ICE.EXE with the /EMM switch in CONFIG.SYS file, you may have to run EMMSETUP.EXE to configure the EMM 4.0 environment. This configuration process allows you to select which portions of memory you would like to make available as EMM 4.0 pages. Running EMMSETUP.EXE is highly recommended if you are using programs that take full advantage of the EMM 4.0 specification.

Default EMM Pages

By default, S-ICE.EXE with the /EMM switch is pre-configured to allow EMM 4.0 pages in the following areas:

- The lower 640K (except for the 1st 64K)
- 64K starting at DDH

You may want to reconfigure for the following reasons:

- You may have a device such as a network that i the D000H area of memory.
- You may want to fill more holes above 640K with EMM pages. This will increase performance and usability of programs like Microsoft Windows. To get maximum performance from Microsoft Windows you should fill every available page with expanded memory.

Customizing the EMM Page Map

To configure the EMM map you must use the utility EMMSETUP.EXE. EMMSETUP.EXE allows the page map to be altered, then modifies S-ICE.EXE with the changes.

EMMSETUP makes its best guess on automatically configuring the EMM map. EMMSETUP will try to fill much of the address space as possible with mappable pages while working around video cards and ROMS. If its guess is not good enough or not to your liking you can override it. Overriding may be necessary if you have a network, a special video adapter or a memory- mapped option adapter.

To configure the EMM map enter:

◊ EMMSETUP file-name

The file-name parameter should be S-ICE.EXE. EMMSETUP can also be used with MagicCV release 3.0, in which case the parameter should be NUMEGA.SYS. This parameter is required because EMMSETUP writes the configuration information directly into the driver file. EMMSETUP now has the option of enabling memory blocks for loading high of device drivers and T&SR programs. You must select this feature on EMMSETUP's initial screen.

EMMSETUP displays a matrix of 16K memory pages available in the lower 1 megabyte region. The matrix is divided into 16 columns each representing 64K (from 0 to 10000H). There are 4 rows representing the four 16K pages in each 64K region.

Each block of the matrix can contain an E, X, R or V. Blocks that contain an E are available as EMM pages; blocks that contain an X are not. Blocks that contain an R are memory areas that have been identified by EMMSETUP as ROM areas. You can override these areas with an E if desired, however, this should only be done if the ROM is never accessed. Blocks that contain V are identified as video memory. We have made worst case assumptions on video memory. Your particular video card may not take up as much as we have 'guessed'. You can override the memory blocks that contain unnecessary V's if desired.

If you are satisfied with EMMSETUP's guesses, press the F10 key and S- ICE.EXE will be modified with these parameters. You must reboot before any changes made to S-ICE.EXE will take effect. If you wish to override EMMSETUP's guesses, do so at this time.

To enable expanded memory you must have 4 and only 4 contiguous F's above 640K. To load high device drivers or T&SRs you must place H's in UN-occupied memory blocks above 640K.

Note: If you want to load high device drivers and T&SR programs, but do not want EMM (expanded) memory, then make sure there are no E's or F's in the memory map.

Including and Excluding Areas from EMM

To include an area as EMM 4.0 memory simply guide the cursor to the desired block, then type E. Conversely, to exclude an area from EMM 4.0 memory, guide the cursor to the block and type X. When you are satisfied with your changes, press F10 to exit the program. All changes are automatically stored in the S-ICE.EXE file. If you wish to exit without modifying S-ICE.EXE press ESC. You must reboot before any changes made to S-ICE.EXE will take effect.

When including upper memory blocks keep in mind the following:

- CGA occupies from B800H to C000H.
- MDA occupies from B000H to B100H.
- Most Hercules cards occupy from B000 to C000H.
- EGA occupies from A000H to C000H and from C000H to C400H.
- VGA (mother board) occupies from A000H to C000H.
- VGA (option card) occupies from A000H to C000H and C000H to C800H.
- PS/2 System ROM occupies from E000H to 10000H.
- PS/2 ESDI ROM occupies from CC00H to D000H
- Most AT Compatible Roms occupy from F000H to 10000H.

- Compaq systems, Micronix motherboard systems, and most Chips and Technologies motherboard systems move the EGA/VGA ROM to E000H However they still occupy the C000H region as well.
- Token Ring Networks usually occupy from CC00H to E000H.
- Many Networks occupy memory regions in the D000H area.

The above guidelines are for 'generic' devices, Many implementations by different computer vendors and adapter card vendors will vary.

Other EMM Features

S-ICE.EXE with the /EMM switch has two features that are automatically enabled depending on your system configuration. These features are backfilling and relocating the page frame.

Increasing Conventional Memory

System memory will automatically be backfilled up to the first non- mappable page. This means it starts looking at contiguous E's at location 1000, and continues until it finds the first non-contiguous E. If the contiguous E's go beyond the amount of your system's base memory, memory will backfilled up to the first R, V, or X that is found.

The benefit of backfilling is that you can increase the amount of usable system memory to greater than 640K. The backfilled memory is available within DOS. If you do not want memory backfilled, use EMMSETUP to make page non-mappable (X) at the point you wish system memory to end.

Note: Monochrome-only systems (MDA) can backfill up to B000H to add an additional 64K to conventional memory CGA systems can be backfilled up to B800, adding an additional 96K to conventional memory. EGA and VGA systems can be backfilled only if no graphics programs will be run. You can backfill an EGA or a VGA system up to B800:0 if no graphics programs will be run.

Warning: If memory is backfilled,DO NOT UNLOAD SoftICE. Doing so will cause your system to crash.

Automatic Page Frame Locating

Most EMM-knowledgeable programs require a 64K page frame that is not used as normal DOS memory. This is normally located above the video device area. However in some systems there is no 64K contiguous region to place the page frame. In these instances S-ICE.EXE 'steals' top 4 mappable pages of lower memory. The net result that lower DOS memory shrinks by 64K.

EMM Debugging

A range break point or a break point on memory that is in an EMM mappable area will stay at that address no matter which EMM page is mapped in.

When debugging EMM programs, the EMMMAP command may also be very useful. *Specialized Debugging Commands* on page 92 for more information.

The D, E, S, F, and C commands can be used to view or modify any allocated EMM handle page. The page does not have to be currently mapped in. The syntax of these commands is similar to that of the commands when being used for non-EMM pages, except for the following:

- In the D, E, S, and F commands, the address portion of the command must be specified in the following way:

  Hhandle# Ppage# offset

  where handle is a number specifying which EMM handle to use, page is a number specifying which EMM page to use, and offset is a number from 0 to 4000H, specifying the offset from the beginning the page.

Example: DB H1 P3 0 This command will dump bytes from page 3 of handle 1, starting at offset 0.

- The C command must be specified in the following way:

  C Hhandle# Ppage# offset1 Llength offset2

  where handle and page are the same as above. offset1 is a number from 0 to 4000H, specifying the offset from the beginning of the page, where the first data block to be compared is located. offset2 is a number from 0 to 4000H, specifying the offset from the beginning of the page, where the second data block to be compared is located. Example: C H2 P4 00 L10 1000 This command will compare the first 10 bytes of memory located at offset 0 of page 4 of handle 2 with the first 10 bytes of memory located at offset 1000 of page 4 of handle 2.

Note: Subsequent uses of the D, E, S, F, and C commands will continue to use the handle and page last specified. To get back to conventional memory, use one of the above commands with a segment specified in the address field, for example:

  ◊ D 0:0

Loading High Of Resident Programs

The LH.EXE utility allows loading certain resident programs into available memory blocks between 640K and I megabyte. Before using LH.EXE you must reserve memory for loading high using EMMSETUP.EXE. This is done by placing an 'H' in each memory block above 640K that you wish to have as a load high area. To load a resident program high enter:

```
LH program-name [program parameters]
```

If there is a high memory block large enough to hold the program, the program will be loaded into it.

If no program-name follows LH on the command line, a memory map is displayed of the DOS loadable device drivers and resident programs loaded high along with available memory.

Note: You can not load all resident programs with LH.EXE. You must experiment to see which programs can be loaded high.

Loading High Of MS-DOS Loadable Device Drivers

The LD.SYS utility allows loading certain MS-DOS loadable device drivers into available memory blocks between 640K and I megabyte. Before using LD.SYS you must reserve memory for loading high using EMMSETUP.EXE. This is done by placing an 'H' in each memory block above 640K that you wish to have as a load high area.

To load an MS-DOS loadable device driver high, you must place the following line in your CONFIG.SYS file:

```
DEVICE = \path\LD.SYS device-name [parameters]
```

path - Path containing LD.SYS

device-name - Name of DOS Loadable device driver including path

If there is a high memory block large enough to hold the device driver, the program will be loaded into it when you boot.

To display a memory map of DOS loadable device drivers and resident programs loaded high use the LH utility with no parameters from DOS.

Note: You can not load all DOS loadable device drivers high. You must experiment to see which drivers can be loaded high. Make sure you have a boot disk handy While experimenting.

Adding High Memory to MS-DOS

The ADDHI.EXE utility allows you to add high memory areas to the DOS pool of free memory. Before using ADDHI.EXE you must reserve memory for adding high using EMMSETUP.EXE. This is done by placing an 'H' in each memory block above 640K that you wish to have as a add high area.

VCPI Support

VCPI (Virtual Control Program Interface) is automatically enabled when you use the /EMM switch on the S-ICE.EXE line in CONFIG.SYS. VCPI support lets you run VCPI applications that use DOS extenders when SoftICE is loaded. It does not allow you to debug these applications in protected mode. VCPI conforming applications include Lotus 123 version 3.0 and Autocad.

VCPI support does NOT enable SoftICE to run with other VCPI control programs, such as Quarterdeck's QEMM and Qualitas's 386MAX.

8 Back Trace Ranges

Introduction

SoftICE can collect instruction information in a back trace history buffer as your program executes. These instructions can then be displayed after a bug has occurred. This allows you to go back and retrace a program's action to determine the actual flow of instructions preceding a break point.

Instruction information is collected on accesses within a specified address range, rather than system wide. The ranges can be from 1 byte to 1 megabyte, so if desired, complete system information can be obtained. Using specific ranges rather than collecting all instructions is useful for two reasons:

1. The back trace history buffer is not cluttered by extraneous information that you are not interested in. For example, you may not be interested in interrupt activity and execution within MSDOS.

2. Back trace ranges degrade system performance while they are active. By limiting the range to an area that you are interested in, you can improve system performance greatly.

SoftICE has two methods of utilizing the instructions in the back trace history buffer:

1. The SHOW command allows you to display instructions from the back trace history buffer. You must specify how many instructions you wish to go back in the buffer.

2. The TRACE command allows you to go back and replay instructions from the back trace history buffer, This way you can see the instruction flow within the context of the surrounding program code or source code.

Using Back Trace Ranges

To use back trace ranges you must do the following:

1. 1. Allocate a back trace history buffer of the desired size by inserting the /TRA switch on the S-ICE.EXE line in CONFIG.SYS. For example, to create a back trace buffer of 100K you might have the following line in your CONFIG.SYS file:

   DEVICE = S-ICE.EXE 100

   A back trace history buffer of 10K is allocated by default. If this is suitable for your needs you do not have to allocate a larger buffer. The history buffer size is only limited by the amount of extended memory available.

2. Enable back trace ranges by creating a memory range break point with the T or TW verb. For example:

   BPR 1000:0 2000:0 T

   The T and TW verbs do not cause break points instead they log instruction information that can be displayed later with the SHOW or TRACE commands.

3. Set any other break points if desired.

4. Exit from SoftICE with the X command.

5. After a break point has occurred, or you have popped SoftICE up with the hot key, you can display instructions in the buffer with the SHOW command. For example, to go back 50 instructions in the buffer and display instructions type:

   SHOW 50

6. To replay a series of instructions you must first enter trace simulation mode with the TRACE command. To begin replaying the sequence of instructions starting back 50 in the buffer type:

   TRACE 50

7 After you have entered trace simulation mode, you can trace through the sequence of instructions by using the XT, XP, or XG commands. This allows you to re-enact the program flow. For example, you can single step through the sequence of instructions in the buffer, starting at the instruction specified by the TRACE command, by typing:

```
 XT
XT
.
.
.
XT
```

The XT command single steps through the back trace history buffer. The XP command program steps through the back trace history buffer. The XG command goes to an address in the back trace history buffer.

8 To exit from trace simulation mode type:

```
 TRACE OFF
```

9 To reset the back trace history buffer, use the X command.

Special Notes

While in trace simulation mode, most SoftICE commands work as normal, including displaying the memory map, and displaying and editing data. The exceptions are:

- Register information is not logged in the back trace history buffer, so the register values do not change as you trace through the buffer, except for CS and IP.
- Commands that normally exit from SoftICE do not work while in trace simulation mode. These are X, T, P, G, EXIT.

As you peruse instructions from the back trace history buffer with the SHOW and TRACE commands, you may notice peculiarities in instruction execution. These are caused by jumps in and out of the specified range. These usually occur at jumps, calls, returns and entry points.

When you have a hang problem or other difficult bug that requires back trace ranges, you must often use very large ranges in order to narrow the scope of the problem. Once you have a better idea of the specific problem area, you go to smaller ranges.

Large back trace ranges are often very slow. When using large ranges you are usually trying to get a general idea where the problem is. SoftICE has a special ‘COARSE’ mode for doing large ranges. This speeds up the ranges a factor of three or more, but limits the amount of instructions in the history buffer.

Coarse mode only collects instructions that do a memory write within the specified range. As you are replaying instructions with trace simulation mode after a ‘coarse’ range you will notice that the flow skips around rather than sequentially executing instructions.

Coarse ranges work best for large ranges and tend to be less effective for small ranges.

To enable a ‘coarse’ back trace range, use the BPR command with the TW verb instead of the T verb. For example:

```
BPR 1000:0 2000:0 TW
```

For further information on back trace ranges see the command descriptions for:

SHOW, TRACE, XT, XP, XG, XRSET, BPR

9 Using SoftICE with MagicCV or MagicCVW

Introduction

MagicCV allows you to run Microsoft's CodeView in less than 8K of conventional memory on your 80386 machine.

MagicCVW allows you to run Microsoft's CodeView for Windows in less than 8K of conventional memory on your 80386 machine.

Using SoftICE in combination with MagicCV or MagicCVW allows you to have the power of SoftICE while still having the convenience of using the CodeView product that you are familiar with.

In the rest of this chapter, statements about MCV will apply to both MagicCV and MagicCVW, and statements about CV will apply to both CodeView and CodeView for Windows.

Running SoftICE with MagicCV or MagicCVW

To use SoftICE 2.0 and MCV together, you must install S-ICE.EXE as a loadable device driver. S-ICE.EXE comes on the SoftICE diskette. S- ICE.EXE replaces NUMEGA.SYS in CONFIG.SYS. Use the /MCV, /EMM, and the /EXT switches as if using MagicCV or MagicCVW alone. There are additional switches that you may want to use for SoftICE. Refer to chapter 6 for information about these switches.

To run MagicCV or MagicCVW after SoftICE has been loaded, refer to your MagicCV or MagicCVW manual.

Note: MagicCVW requires SoftICE version 2.00 or greater. MagicCV requires SoftICE version 1.02 or greater. The S-ICE.SYS and NUMEGA.SYS drivers were shipped with some versions of SoftICE. The S-ICE and NUMEGA drivers must be replaced by S-ICE.EXE before you can run MagicCV and Soft- ICE 2.0 together.

Special Considerations

Two Virtual Machines

When you are using both SoftICE and MCV together, you must keep in mind that CV is in a separate virtual machine from the target environment. You can pop SoftICE up from either virtual machine, i.e., when CV is running, or when the target program is running.

If you pop SoftICE up while the target program is running everything works as defined in the SoftICE manual. If you pop SoftICE up while CV is running (typically done to break points), you must keep a few points in mind:

- The registers are those of CV and they CAN NOT be changed.
- For convenience, the SoftICE MAP command displays the memory map of the target program virtual machine, not the memory map of the CV virtual machine. The highlighted area in the memory map may not be correct.
- Any display or modification of memory occurs in the target program's virtual machine.
- You have no visibility into the CV virtual machine except for the display of register values. Remember that when popping up the SoftICE window while CV is active, the register values are those of CV and should not be modified.
- Instruction and program tracing is disabled from the SoftICE window when CV is active. This is to prevent confusion, because a trace would actually step through CV, not through the target program.
- If you attempt to do a SoftICE Trace (T) or Program Step (P) command while CV is active, you will get the warning message: "Function not available in CV virtual machine." To trace through your target program code instead, you can do one of two options:
- Use the CV trace command. To do this, exit the SoftICE window using the SoftICE X command, then do one or more CV traces to step through the target program.
- Use SoftICE to go to the target program address, then use the SoftICE T or P commands to step through your target program. To do this, exit the SoftICE window with the SoftICE X command, then press the 'F3' key until CV is in 'mixed mode'. This allows you to see both the source lines and the instruction addresses. Pop up SoftICE. If the SoftICE window is not already in narrow mode, use the SoftICE WIN command to change the window size. Move the SoftICE window so you can see the instruction

addresses on the left side of the screen. Now you can use the SoftICE G command to go to one of the addresses. Be sure to type in the full address, including the segment and the offset. Then enter 'G' in the CV window. At this point, CV is not active, so you can use the SoftICE T or P commands to step through t target program.

CodeView's SHELL command

If you run the DOS shell from within the CodeView virtual machine, the DOS shell is part of the virtual machine. Because of this, you should not run any TSRs when you are in the DOS shell. If you do, when you exit CodeView the TSRs will disappear along with the virtual machine. This is dangerous, because any interrupt vectors that were not restored could hang your machine.

CV's /R switch

SoftICE takes advantage of many of the 80386 features including the 80386 debug registers. This means that the debug registers are not available for CV, so you cannot use the CV /R switch when running with SoftICE. If you do use the /R switch, SoftICE gives you a general protection error. At this point, you can press "C" to continue, then rerun CV without the /R switch, and use the SoftICE break points.

The CV /R switch works when you are running MCV without SoftICE.

The SoftICE ACTION Command

The ACTION command allows three different methods activating CV from a SoftICE break point. The best choice of action is ACTION NMI. If you experience any problems with ACTION set to NMI (usually because an adapter card in your system is using NMI), use ACTION INT1.

10 Using SoftICE with BoundsChecker

BoundsChecker gives you the protection of a protected mode operating system under MS-DOS. When your program is running, BoundsChecker protects your program's CODE and all memory outside your program. When an MS-DOS system call or BIOS call or interrupt occurs, BoundsChecker prevents the system software from corrupting your program. So BoundsChecker can not only detect problems caused by your program, it can also determine if a T&SR or other program is clobbering you.

Each time you make a change to your program, run BoundsChecker while testing the new code. Your program runs at full speed, and if you accidentally access out-of-bounds memory, BoundsChecker pops up displaying the offending source line.

Using SoftICE in combination with BoundsChecker is very useful when the bug found by BoundsChecker is not clearly self-explanatory. You may need to use SoftICE to look at data, to debug a little, or to rerun the program with SoftICE's back trace capability to determine why the out-of-bounds access occurred.

Loading BoundsChecker to Use with SoftICE

To use BoundsChecker for DOS with SoftICE for DOS, you must first:

1. Install BoundsChecker on your hard disk using the BoundsChecker installation program.

2. Replace the DEVICE=d:\path\BC.SYS line in your CONFIG.SYS file with DEVICE=d:\path\S-ICE.EXE. Use the same parameters that were on the BC.SYS command line. In addition, you may want to use the /TRA nnnn parameter to create a back trace buffer larger than 10K. You may also need to increase the size of your /SYM nnnn parameter to allow your source and your symbols to be loaded.

Note: You do not need the /BC switch on the DEVICE=d:/path /S-ICE.EXE line in CONFIG.SYS as the BoundsChecker manual states. You must have SoftICE version 2.8 or greater and BoundsChecker version 1.1 or greater for them to coexist.

Running SoftICE with BoundsChecker

Run BoundsChecker. When BoundsChecker pops up, if you want to enter SoftICE to do further debugging, select Options on the main menu, then select SoftICE. To re-enter BoundsChecker, simply exit SoftICE with the hot key sequence or the X command.

If you don't have enough extended memory to run BoundsChecker, you can save space by running BoundsChecker with option /S in this form:

```
BC /S program-name
```

This stops source from loading up into extended memory for use by SoftICE. The disadvantage is that SoftICE will show line numbers, but will not show source code.

Note: SoftICE range break points and back trace ranges will be disabled while the BoundsChecker is running.

The BOUNDS Command

The new command, BOUNDS, is used for turning bounds checking on and off from within SoftICE. This is useful if you want to stop to do some debugging from within a BoundsChecker session, then return to bounds checking after you have debugged a portion of the program.

The syntax of the BOUNDS command is:

```
BOUNDS [ON | Off]
```

BOUNDS OFF turns off bounds checking, and BOUNDS ON turns bounds checking back on. If no parameters are specified, then the current state is displayed.

11 Advanced Features

Using SoftICE with other Debuggers

SoftICE was designed to work well with other debuggers. Each debugger offers different features, and therefore can require special treatment. This section will describe some ways to use several debuggers effectively.

Debuggers that Use DOS

Many debuggers use DOS and ROM BIOS to perform their display and keyboard I/O. Special consideration must be taken when using these debuggers with SoftICE (e.g., DEBUG, SYMDEB, and CODEVIEW), because DOS and ROM BIOS are not fully re-entrant. If a break point occurs while code is executing in DOS or BIOS, a re-entrancy problem can occur.

SoftICE provides optional re-entrancy warning, which is activated with the WARN command. When WARN mode is on, SoftICE checks for DOS or ROM BIOS re-entrancy before generating the ACTION that wakes up the host debugger. When a re-entrancy problem is detected, SoftICE displays a warning message and offers you the choice of continuing to execute the code or returning to SoftICE.

Note that SoftICE itself does not use DOS or ROM BIOS calls in its debugging commands. This means that you can use SoftICE any time, without the worry of re-entrancy problems.

For more information on the WARN command, see *WARN* on page 82.

ACTION Command with other Debuggers

Different debuggers use different methods of activation For a description of these methods see *Activating Other Debuggers* on page 193.

If you want to return to your debugger after a break point reached, you must change the ACTIONto work with your debugger.

In most cases, the action that should be taken after a break point is reached is INT3. For instance, DEBUG and SYMDEB will work best with ACTION set to INT3.

If INT3 doesn't work with your debugger, try INT1 or NMI. CODEVIEW works best with ACTION set to NMI.

Special Considerations

When a break point is set, you must be careful not to set off the break point unintentionally. For instance, if you set a memory break point at 0:0, then use your debugger to dump memory location 0:0, SoftICE will be triggered. If ACTION is set to go to your debugger, then your debugger will be triggered by itself. Since some debuggers cannot be re-entrant, this could be a fatal problem. This problem can also occur with other debugging functions, such as editing or unassembling.

For this reason, it is a good practice to disable the SoftICE break points once SoftICE has helped you get to the point where you want to look around with your debugger.

Using SoftICE with CODEVIEW

SoftICE works best with CODEVIEW when CODEVIEW is either in Assembler mode or Mixed mode. When CODEVIEW is in Source mode with higher-level languages it does not always break correctly. It is always best to use ACTION NMI when you want SoftICE to wake up CODEVIEW.

Debuggers that Use 80386 Break Point Registers

The 80386 has 4 break point registers that are available for use by debuggers. SoftICE uses these for its memory byte, word and double word break points. If the debugger you are using SoftICE with uses these debug registers there will be a conflict. There are two ways to handle this problem.

1. Disable the use of 80386 break point registers in the debugger you are using SoftICE with. Check the documentation of your other debugger for a description of how to do this.

2. Some debuggers automatically use the break point registers if they detect an 80386 processor with no method of turning them off (some versions of SYMDEB do this). For these debuggers do the following:
   - ◊ Bring up the SoftICE window before you start the other debugger.
   - ◊ Turn on SoftICE's break mode with the BREAK command (you may want to do this in the INIT statement of S-ICE.DAT if you are doing this frequently).
   - ◊ Start up your other debugger.
   - ◊ You may now pop up the SoftICE window and turn the SoftICE break mode off if desired.

User-Qualified Break Points

Occasionally you may have the need for a very specific set of break point conditions. If the special conditions require qualifying register values or memory values, you can write a break point qualification routine.

SoftICE contains a very general mechanism for calling user-written break point qualification routines: the ACTION command. When you use the ACTION command, SoftICE can route all break points through special interrupt vector. However, before break points can be routed, the qualification routine must be placed in memory, and the interrupt vector must be pointing to the qualification routine.

All registers are identical to the values when the SoftICE break point occurred. It is the responsibility of the qualification routine to save and restore the registers. If your qualification routine detects a match of break point conditions, it can do a variety of activities. Some examples of useful activities that a routine can do when a match is found are:

- Store information for later
- send the information directly to a printer or serial terminal
- issue an INT 3 instruction to bring up SoftICE The command 13HERE must be turned on in order for the INT 3 to bring up SoftICE see *Debug Mode Commands* on page 80.

If conditions do not match, the qualification routine in should execute an IRET instruction. To summarize:

1. Create a break point qualification routine in your code space, or anywhere in free memory. The routine must preserve registers. After comparing the desired conditions, the routine can execute either an INT 3 to bring up SoftICE, or an IRET to continue.

2. Point an unused interrupt vector to your qualification routine. This can be done either within your code or from SoftICE.

3. In SoftICE, set ACTION to the interrupt- number that was used to point to your qualification routine.

4. In SoftICE, set 13HERE on. This is necessary to bring up SoftICE after the conditions have been met.

5. Set the SoftICE general break point conditions. When any of these break point conditions are met, your qualification routine will be called.

Example of a User-Qualified Break Point

This section contains an example of a user-qualified break point that compares for the conditions of U = 3, BX = 4 and CX = 5 when a break point goes off.

First, we create the qualification routine. For the purposes of this example, we will assemble the command directly into memory with the Soft- ICE interactive assembler. For this example we will arbitrarily assemble the routine at location 9000:0H. The following statements are entered into SoftICE:

```
 A 9000:0
9000:0 CMP AX,3
9000:3 JNE 10
9000:5 CMP BX,4
9000:7 JNE 10
9000:A CMP CX,5
9000:D JNE 10
9000:F INT3
9000:10 IRET
```

Now that the routine is in memory, you must point an interrupt vector to the routine. For this example, we arbitrarily pick INT 99H. To place 9000:0H in the INT 99H vector enter:

- ED 0:99*4 9000:0

Set the ACTION command so that SoftICE will call your break point qualification routine on every break point.

- ACTION 99

Set 13HERE on so the qualification routine can activate SoftICE when the conditions occur.

- 13HERE ON

Now you need to set the break points. For this example, we are just interested when the registers are: U = 3, BX = 4, CX = 5 in a specific program, and we do not want any further qualification. To do this, use a range break point on memory read:

- BPR segment:starting-offset segment:ending-offset

This will cause your break point qualification routine to be called after every instruction is executed in the specified memory range. When the register conditions do not match, then the IRET instruction is executed. When the conditions finally match the specified qualifications, the INT 3 is executed and SoftICE is popped up.

When SoftICE pops up, the instruction pointer will be pointing at the INT3 in your qualification routine (9OOO:FH in our example). To get to the instruction after the one that caused the break point, you must change the instruction pointer to point to the IRET instruction (F000: 10H in the example) and single step one time. This is accomplished with the following SoftICE commands

- RIP IP + 1
- T

After your break conditions have gone off, remember to change the ACTION command back to ACTION HERE that subsequent break points do not go through your qualification routine.

The Window in Graphics Mode

The screen is switched to text mode when SoftICE is invoked. If the screen was in graphics mode or 40-column mode, the graphics display is not visible while the window is up. For users who must see the graphics display while debugging, three features are provided. The first feature allows the SoftICE window to display on a second monitor (see the ALTSCR command). The second feature allows you to restore the screen while you are doing P or T instruction step commands (see the FLASH command). The third feature allows you to restore the program screen temporarily (see the RS command).

If SoftICE does not seem to be following your program into graphics mode, try turning WATCHV on (see *Screen Control Commands* on page 121 for details).

Expanded Memory Debugging Features

A range break point or a break point on memory that is set in an EMM mappable area will stay at that address no matter which EMM page is mapped in.

When debugging EMM programs, the EMMMAP command may also be very useful. See *Specialized Debugging Commands* on page 92 for more information.

The D, E, S, F, and C commands can be used to view or modify any allocated EMM handle page. The page does not have to be currently mapped in. The syntax of these commands is similar to that of the commands when being used for non-EMM pages, except for the following:

- In the D, E, S, and F commands, the address portion of the command must be specified in the following way:

  Hhandle# Ppage#

  offset where handle is a number specifying which EMM handle to use, page is a number specifying which EMM page to use, and offset is a number from 0 to 4000H, specifying the offset from the beginning of the page.

Example: DB H1 P3 0

This command will dump bytes from page 3 of handle 1, starting at offset 0.

- *The C command must be specified in the following way:

  C Hhandle# Ppage# offset1 L length offset2

  where handle and page are the same as above. offset1 is a number from 0 to 4000H, specifying the offset from the beginning of the page, where the first data block to be compared is located. offset2 is a number from 0 to 4000H, specifying the offset from the beginning of the page, where the second data block to be compared is located.

Example: C H2 P4 00 L10 1000

This command will compare the first 10 bytes of memory located at offset 0 of page 4 of handle 2 with the first 10 bytes of memory located at offset 1000 of page 4 of handle 2.

Note: Subsequent uses of the D, E, S, F, and C commands will continue to use the handle and page last specified. To get back to conventional memory, use one of the above commands with a segment specified in the address field, for example:

◊ D 0:0

Extended Memory Debugging Features

The D, E, S, F, and C commands can be used to view or modify extended memory. Extended memory reserved by SoftICE can not be displayed. The syntax of these commands is similar to that of the commands when being used for conventional memory:

- In the D, E, S, and F commands, the address portion of the command must be specified in the following way:

  M megabyte address

  where megabyte is a number specifying which megabyte to use, and address specifies the address in the specified megabyte. Example: DB M 2 0:0 This command will dump bytes from start of the megabyte starting at linear address 200000H.

- The C command must be specified in the following way:

  C M megabyte address1 L length address2

  where megabyte and address1 are the same as above. address2 specifies the address in the specified megabyte, where the second data block to be compared is located.

Example: C M 3 1000:2000 L10 3000:4000

This command will compare the first 10 bytes of memory located at 1000:2000 with the first 10 bytes of memory located at 3000:4000.

Note: Subsequent uses of the D, E, S, F, and C commands will continue to use the last megabyte specified. To get back to megabyte 0 (conventional memory), use one of the above commands with 0 specified as the megabyte, for example:

◊ D M 0

Remote Debugging

SoftICE is capable of displaying all of the information from the command window over a serial port. The hot key is still activated via the system keyboard but once SoftICE is popped up, both the system and the remote keyboard will be active. To activate remote debugging use the following sequence:

1. Set the BAUD rate with the DOS MODE command to the same baud rate as the remote terminal.

2. Within SoftICE, set PRN to the correct serial port.
   Example: `PRN COM1`

3. Within SoftICE, enter SERIAL ON. At this point, you may enter information on either keyboard, and the command window output will go to both screens.

4. You will probably want to get rid of your Code, Data and Register windows as these will not be displayed across to the remote terminal.

5. If you do not want the SoftICE screen up on the host machine then turn ALTSCR ON from within SoftICE.

CONFIG.SYS Editor

CONFIG EDIT (CE.EXE) is an on-the-fly text editor for CONFIG.SYS. CONFIG EDIT is useful if you have to make occasional changes to your CONFIG.SYS. It is especially useful if you suspect that a driver in CONFIG.SYS may hang the system. It is advisable to use CONFIG EDIT when installing SoftICE in your CONFIG.SYS for the first time. Install CONFIG EDIT by placing CE.EXE as the first DEVICE = line in your CONFIG.SYS file. For example:

```
DEVICE = /S-ICE /CE.EXE
```

When your system boots, you will hear a tone. After the tone, you have a short time to press any key. If you press a key CE will take over and allow you to edit CONFIG.SYS.

When you have edited your CONFIG.SYS file, you may exit CE by pressing one of the following keys:

- F1 Pressing F1 exits and changes CONFIG.SYS for this boot only. The changes are not permanent.
- F10 Pressing F10 exits and changes CONFIG.SYS for this boot and subsequent boots.
- ESCPressing ESC exits with no changes

CE can also be run From the DOS command line. This is for a quick look or quick changes to CONFIG.SYS. Simply enter CE from the DOS command line. The /Q switch (Quiet) will disable the initial sound made by CE. when it is installed in CONFIG.SYS.

Back Door Commands

SoftICE contains commands for controlling SoftICE from an MS-DOS program. A program can take advantage of powerful break points for special debugging jobs or hardware simulation projects.

These calls all have the following calling sequence:

```
MOV AH,09
MOV AL,SUB-FUNCTION
MOV SI,'FG'
MOV DI,'JM'
INT 3
```

The sub-functions are available:

AL value	Description
10H	Display information in the SoftICE window.
11H	Do a SoftICE command.
12H	Get break point information.
13H	Set SoftICE break point.
14H	Remove a SoftICE break point.

The following paragraphs give more detailed information about these subfunctions.

AL = 10H --Display Information In the SoftICE window.

This is useful for diagnostic writes - especially from within interrupt routines and other areas that may have reentrancy concerns.

Input: DS:DX - > Zstring of text characters to be displayed

The Zstring can be a maximum of 100 characters and can contain carriage returns (0DH).

AL = 11H -- Do a SoftICE command.

This allows you to generate a SoftICE command from your program. This is used for all non-break point commands. To set SoftICE break points from your program see AL = 13H below.

Input: DS:DX - > Zstring that contains a SoftICE command.

The Zstring can be a maximum of 100 characters. Each SoftICE command in the string should end with a carriage return (0DH).

AL = 12H -- Get break point Information.

Returns the break point number of the last break point set and the last break point that went off.

This is useful when setting break points from hardware control or doing hardware simulation.

Returns:DH - entry number of last break point that went off

DL - type of last break point that went off

BH - entry number of last break point set

BL - type of last break point set

The entry number is the same as is displayed in the BL command.

The types are:0 - BPM (break point register types)

1 - I/O

2 - INT

3 - BPX (int 3 style BP)

4 - Reserved

5 - Range

AL = 13H -- Set SoftICE break point.

Use this command to set SoftICE break points from program control.

Input: DS:DX - pointer to break point structure

Returns: ax = error code

bx = break point number

Very little parameter value checking is done, but the following ; errors are returned.

```
OK EQU 0
BP_TABLE_FULLEQU 3
MEM_LlM_ERREQU 6
IO_LlM_ERREQU 7
RANGE_LlM_ERREQU 9
DUP_ERREQU l6 ;duplicate break point
Break point structure
bp_entrystruc
    bp_typedb ?
    bp_addrldd ?
    bp_addr2dd ?
    bp_addr3dd ?
    bp_modedb ?
    bp_mode2db ?
    bp_sizedb ?
    bp_cntdb ?
```

```
    bp_statedb ?
bp_entryends
```

The following break point types are allowed:

```
MEM_LOCequ ;Memory location break point (BPM).
 MEM_RANGEequ 1;Memory range break point (BPR).
 IO  equ 3;I/0 break point (BPIO).
 INT_BPequ 4;Interrupt break point (BPINT).
 X_BPequ 5;Execution break point (BPX).
```

Here are the possible break point modes and sizes.

```
Break point modes
READ_MODEequ 01
WRITE_MODEequ 02
EX_MODEequ 04
Break point sizes
BYTequ 0
WRDequ 1
DBLequ 3
```

The following paragraphs give information on how to fill the break point structure for each break point type.

Setting memory location break points

bp_type = MEM_LOC

bp_addr1 = address of break point

bp_mode = one of following:

READ_MODE

WRITE_MODE

EX_MODE or WRITE_MODE

EX,MODE (execute break point)

bp,size = one of following:

BYT

WRD

DBL

bp,cnt = Number of instances before breakpoint occurs

All unused fields should be 0.

Setting memory range break points

bp_type = MEM_RANGE

bp_addrl = lower range limit

bp_addr2 = upper range limit

bp_mode = one of following:

READ_MODE

WRITE_MODE

READ_MODE or WRITE_MODE

bp_cnt = Number of instances before breakpoint occurs

All unused fields should be 0.

Setting I/O break points

bp_type = I/O

word ptr bp_addr1 = I/O address

bp_mode = one of following:

READ_MODE

WRITE_MODE

READ_MODE or WRITE_MODE

bp_cnt = Number of instances before breakpoint occurs

All unused fields should be 0.

Setting interrupt break points

bp_type = INT_BP

bp,addr1 = Interrupt #

bp,addr2 = Optional value to check

bp,mode = register to check

0 - no value checking

1 - check AL

2 - check AH

3 - check AX

Setting execution break points

bp_type = X_BP

bp,addr1 = address of break point

bp,addr2 = overlay number (0 = root)

AL = 14H -- Remove SoftICE break point.

Input:BX = Break point number

Returns:BX = ??? when set

12 Special Debugging Problems

SoftICE can be a powerful tool in stand-alone mode. This chapter describes techniques for debugging system-level components using SoftICE in stand-alone mode. When using SoftICE as a stand-alone debugger, the ACTION must be set to HERE.

Loadable Device Drivers

Debugging DOS loadable device drivers requires a debugger that does not make DOS calls. SoftICE can be used in stand-alone mode if your debugger uses DOS.

There are two methods for debugging loadable device drivers:

1. Use the MAP command to find the location of your loadable driver. Display the device driver header to find the strategy or interrupt entry point. Setting a break point at the entry to strategy or interrupt will give you control within the device driver. Single step, or set break points further on, to continue debugging. Debugging the device driver initialization code requires resetting the system with the BOOT command. Use the technique stated above to set a break point within the driver code. The BOOT command will retain SoftICE and break points.

2. The second method requires placing special code in your driver. Do this with the 13HERE ON command (see *Debug Mode Commands* on page 80). Place an INT 3 opcode (CCH) in your device driver at the point where control is desired. When the INT 3 executes, control comes to SoftICE. You can then use an RIP command to set the instruction pointer to get around the INT 3.

If you wish to debug your initialization sequence, make sure that SoftICE is loaded in CONFIG.SYS prior to the driver you are trying to debug. Place the 13HERE ON command in the INIT string in SoftICE.DAT. With this method you do not have to use the BOOT command.

If you are debugging your device driver symbolically or with source you must load the symbol file and the source files separately from the device driver. The symbol file and source files are loaded with the SoftICE program loader LDR.EXE. When LDR.EXE is used to load only the symbols and source you must use it in the form:

LDR file-name.SYM

The extension of the symbol file must be specified. See *Loading Programs and Symbol Files* on page 151 for more details about LDR.EXE.

After loading the symbol file and source files with LDR.EXE you must enter SoftICE and relocate the symbols relative to the start of your device driver. Symbols are relocated with the SoftICE SYMLOC command. The syntax of the SYMLOC command is:

SYMLOC segment

The segment value is obtained from the MAP command. See the description of the SYMLOC command for more details.

Boot Loaders

Debugging boot loaders or self-booting programs requires using SoftICE as a stand-alone debugger. You must first boot into DOS and load SoftICE. The easiest method of debugging boot loaders is to set a break point at a known address within the boot loader, and then use the BOOT command to reset the system. SoftICE is retained throughout the boot process with the break points still set. If a known address is difficult to find an execution break point can be set at 7C0:0H before the BOOT command. This is the address where the ROM BIOS loads the boot sector into memory.

Another method requires turning 13HERE mode on (see *Debug Mode Commands* on page 80). Place an INT 3 opcode (CCH) in your program at the point where control is desired. When the INT 3 executes, control comes to SoftICE, You may also use both symbols and source debugging while debugging a boot loader. See the SYMLOC command for more information on how to relocate your symbols and source to the segment where your boot loader has been loaded

Interrupt Routines

SoftICE allows break points and single stepping within hardware interrupt service routines (timer, keyboard, etc.).

Single stepping and setting break points in interrupt service routines is allowed with SoftICE. You can even single step through the keyboard interrupt routine while SoftICE is using the keyboard for input.

In most cases, SoftICE must be used as a stand-alone debugger when debugging interrupt service routines. To set a break point on the address of the interrupt service routine, use one of the following methods:

1. Use the display double command:

   DD interrupt-number * 4 L 1

   The address displayed is the address of the first instruction of the interrupt service routine. Set a execution break point on this address.

2. Use the command:

   BPINT interrupt-number

Non-DOS Operating Systems

Non-DOS real address mode operating systems can be debugged with SoftICE. If the operating system is not very DOS compatible you may have to load SoftICE under DOS, and then use the BOOT command to start the non-DOS operating system. Follow the instructions for debugging boot sequences and self-booting programs explained in section 12.2.

The MAP and WARN commands may not function properly under a non-DOS operating system, but break points and the other debugging commands will work correctly.

If debugging with symbols or source you must load symbol files and source files while still under DOS or in the DOS compatible mode of your operating system.

13 Theory of Operation

Activating Other Debuggers

SoftICE works with most other debuggers by taking advantage of the 8086 family break point interrupt (INT 3). Most debuggers use the single byte INT 3 (CCH) instruction to produce break points. The target instruction is replaced by an INT 3. When the target address is executed, control is given to the debugger's INT 3 handler. The debugger then replaces the (CCH) with the first byte of the original instruction.

When SoftICE break points occur, one of several events can happen, depending on the ACTION command. Typically, when using SoftICE with another debugger, ACTION is set to INT3. When break point conditions match, SoftICE passes control to the host debugger by simulating an INT 3.

Some debuggers may not work properly by simulating INT 3's. For these debuggers, two other ACTION options are provided. They are INT1 and NMI. IX 1 is the 8086 family single-step interrupt. Most debuggers will handle an unsolicited INT 1 as a break point. NMI is supported by many debuggers as a means of breaking out of a hung condition. These debuggers were designed for hardware break-out switches that produced non-maskable interrupts. When ACTION is set for NMI, SoftICE simulates the non- maskable interrupt (Interrupt 2). CODEVIEW works best with ACTION set to NMI.

Virtual Machine Basics

The magic of SoftICE is made possible by the virtual machine capability of the 80386 processor. SoftICE runs in the 80386 protected mode and manages the DOS environment. The 80386 protection circuitry gives SoftICE complete control of the DOS environment while protecting it from a wayward program.

How are SoftICE break points generated?

SoftICE uses three different 80386 features to produce break points:

- Break points on memory location use the 80386 break registers
- Break points on memory ranges use the 80386 paging mechanism
- Break points on I/O instructions use the I/O privilege level and I/O bit mask

How is the BREAK command implemented?

The BREAK command allows use of the keyboard to bring up SoftICE, even when interrupts are disabled and the system is hung. SoftICE virtualizes the interrupt mechanism so that interrupts are never disabled to SoftICE, even when they are disabled to the DOS program running in the virtual machine.

When in break mode, the following instructions are virtualized to make sure the interrupt flag is never cleared:

- PUSHF
- POPF
- STI
- CLI
- INT n
- IRET

Special considerations with virtual 8086 mode

SoftICE runs DOS in an 8086 virtual machine. This capability is a major feature of the 80386 microprocessor. When running real address mode software (DOS, etc.) in a virtual machine some 8086 features must be emulated by a program that controls the virtual machine. In our case, SoftICE controls the virtual machine. The following peculiarities are handled by SoftICE:

- ROM BIOS interrupt 15H functions 87H, 88H, and 89H
- The undocumented loadall instruction
- Address line 20H control
- 80286 and 80386 protected instructions
- 80386 bugs

ROM BIOS interrupt 15H functions 87H, 88H, and 89H

BIOS function 87H allows a program to access memory above one megabyte in the IBM AT or Personal Series 11 architectures through a block move mechanism. Function 88H returns the extended memory size. These functions are used by the VDISK device driver. SoftICE emulates these BIOS calls for VDISK compatibility. Function 89H is normally used to put you into protected mode, but SoftICE can not allow this to happen. Instead it returns with the carry flag set.

The undocumented loadall instruction

The 80286 contains an undocumented instruction called loadall. This instruction was originally placed on the chip for diagnostic purposes and is not generally used by software. However, it is used by some versions of Microsoft's RAMDRIVE which is sold with Microsoft Windows and MSDOS 3.2. SoftICE emulates loadall to the extent of getting RAMDRIVE to work, however it is impossible to do a complete emulation of this instruction.

Address line 20H control

The IBM AT introduced a special feature that allowed some old programs that were originally written for CP/M to function on the 80286 processor. This feature allowed memory accesses that wrapped from the one megabyte region to the zero region on the 8086 to work on the 80286. Some programs disable this 'wrap compatibility' to access memory just above one megabyte in real address mode. SoftICE emulates this ability. This is supported on all 80386 AT machines through the keyboard controller, and through I/O port 92H on the PS/2.

80286 and 80386 protected instructions

Some AT specific programs have used 80286 protected instructions. With the emergence of the 80386, some 80386 programs use 80386 protected instructions. These programs will not work with SoftICE.

SoftICE supports the standard real-address mode extensions that Intel had included with the 80186 & 80286 processors (PUSHALL, POPALL, etc.), but not protected mode instructions such as LGDT, LMSW, etc.

80386 Bugs

There are several 80386 bugs up through the C stepping of the chip. Most of these bugs only apply to protected mode software (such as SoftICE).

A Functional Command List

B ALPHABETICAL COMMAND LIST

C Keystroke Function List

KEYSTROKE FUNCTION LIST

Keystroke Description

Moving the SoftICE window

CTRL ? Move window one row up

CTRL ? Move window one row down

CTRL Move window one row right

CTRL ? Move window one row left

Resizing the SoftICE window

ALT ? Expand the window

CTRL ? Shrink the window

Editing the Command Line

Move the cursor to the right

? Move the cursor to the left

INS Toggle insert mode

DEL Delete current character

HOME Move cursor to the start of the line

END Move cursor to the end of the line

? Display the previous command

? Display the next command

SHIFT ? Scroll one line up in display

SHIFT ? Scroll one line down in display

PAGE UP Scroll one page up in display

PAGE DN Scroll one page down in display

BKSP Delete previous character

ESC Cancel current command

D Error Messages and Descriptions

This appendix lists and explains the error messages that can be generated by SoftICE.

A General Protection Violation Has Occurred. This is typically caused by a protected mode instruction. CS:IP = XXXX:XXXX Type 'C' to Continue Type 'R' to Return to SoftICE.

This message can occur either when an 80386 protected mode instruction is encountered or if there is a segment wrap-around condition. You can often determine the reason for this message by un-assembling the instruction at the specified address. If the first byte of the instruction is an 0FH, then it is probably a protected mode instruction. If the instruction is accessing a word at offset 0FFFFH in a segment then it is a segment wrap problem. If you type C to continue, then control is given to the interrupt 0 handler in the DOS virtual machine.

This message often occurs when a program jumps to an address that does not contain valid code or when valid code has been overwritten.

Attempt To Divide By 0

This message is displayed when SoftICE evaluates an expression and the divisor in a divide operation is zero.

BPM Break Point Limit Exceeded

SoftICE allows a maximum of 4 memory break points. This message is displayed ifyou attempt to exceed the maximum limit.

Break Point Table Full

SoftICE allows a maximum of 16 break points. This message is displayed if you attempt to exceed the maximum limit.

Count Too Large

The SoftICE break point commands allow an optional count field. This field can contain a maximum value of FFH. This error message is displayed if the count value specified is greater than FFH.

DOS Memory Structures Corrupted

This message is displayed if SoftICE detects a problem with the DOS memory block chain when using the MAP command. This message can also occur if you use the MAP command with a non-DOS operating system.

Duplicate Break Point

When a break point is entered, SoftICE compares the break point conditions with those of break points that had been set previously. If the conditions match, this message is displayed.

Interrupt Break Point Limit xceeded

SoftICE allows a maximum of 10 interrupt break points. This message is displayed if you attempt to exceed the maximum limit.

Invalid Opcode Has Occurred CS:IP =XXXX:XXXX Type 'C' to Continue Type 'R' to Return to SoftICE.

When the 80386 encounters an instruction that is illegal, it generates an interrupt 6. SoftICE displays this message and gives you the opportunity to continue or to return to SoftICE. If you type C to continue, then control is given to the interrupt 6 handler in the DOS virtual machine. This message often occurs when a program jumps to an address that does not contain valid code or when valid code has been overwritten.

I/O Break Point Limit Exceeded

SoftICE allows a maximum of 10 I/O break points. This message is displayed if you attempt to exceed the maximum limit.

No Alternate Screen

This message is displayed if the ALTSCR command is used and SoftICE detects only one video adapter.

Parameter is Wrong Size

Certain fields require a specific data type size (byte, word or double word). This message is displayed if the data type size is exceeded. For example, if you use the command 'BPMB 2000:2000 EQ 1234' you are asking SoftICE to look for a byte access at location 2000:2000 with a value of 1234H. Since 1234H is larger than a byte, the command causes this error message to occur.

Parameters Required

Most SoftICE commands require one or more parameters. If a command is entered without the required number of parameters, this message is displayed.

Range Break Point Limit Exceeded

SoftICE allows a maximum of 10 memory range break points. This message is displayed if you attempt to exceed the maximum limit.

Second Parameter Must Be Greater than First

When specifying a memory range, the first number entered must be the lower limit of the range, otherwise this message is displayed.

Segment:Offset Can Not Wrap

Most SoftICE commands do not allow a memory pointer (segment:offset) to wrap from high memory to low. For example, the memory pointer FFFF:FFFF wraps and is illegal. This message is displayed if you attempt to wrap from high memory to low.

SoftICE cannot be loaded. Needs to load at top of memory. Load before any TSR's or control programs.

SoftICE needs to load itself at the highest memory location possible. This memory is then 'mapped out', making it invisible to DOS programs, so they can't crash SoftICE. This message is displayed if SoftICE detects that another program has already been loaded at the top of memory.

SoftICE cannot run with other 80386 control programs

The 80386 only allows ore protected mode program a time, so SoftICE can not coexist with other control programs. When debugging a program that use EMS and EEMS, you could get this error message when you try to load SoftICE, because some 80386 systems come with a control program that uses the 80386 paging system to give you EMS and EEMS with a board that only has extended memory. You can, however, use a true expanded memory board to debug programs that use EMS and EEMS.

SoftICE has already been loaded

This message occurs if you attempt to load SoftICE twice.

SoftICE has not been loaded

This message occurs if you attempt to unload SoftICE when it has not yet been loaded.

SoftICE loads at the top of extended memory. This may conflict with other programs that use extended memory. If you are sure it will not conflict, then answer 'Y', otherwise answer 'N' and refer to the chapter on loading SoftICE with extended memory.

This message occurs if you attempt to load SoftICE into extended memory, and S-ICE.SYS was not loaded in your CONFIG.SYS file. This warning is given to insure that you do not unintentionally wipe out a virtual disk or another program that may be loaded in extended memory. For more information, refer to section 2.2, "Loading SoftICE" and chapter 6, "Initialization Options".

SoftICE will only run on 80386 based machines

SoftICE requires Intel's 80386 microprocessor.

Syntax Error

This message is displayed if the information that was entered did not fit within the structure of any SoftICE command.

The P & G Commands Function In RAM Only

SoftICE uses two methods to implement the P and G commands. The first method uses the 80386 break point registers. However, if you have already set 4 BPM-style break points, SoftICE uses the INT 3 method, which will only work in RAM. If you attempt to use the P or G commands in ROM at this point, SoftICE detects this condition and displays this error message.

Valid Verbs are R, W, RW, X

This message is displayed if an invalid verb is specified in a BPM command. When using the BPM command, the valid choices for verbs are R(read), W(write), RW(read/write), and X(execute).

E Troubleshooting Guide

This appendix gives solutions to some possible problems that you could encounter when using SoftICE. If you do not find the problem here, check the README.SI file on your distribution diskette for any troubleshooting hints that may not have made it into this manual.

Time does not show the correct time at the end of the day.

SoftICE does not let any interrupts go through to the system when the SoftICE window is up. This does not affect the real time clock at all, so the next time you reboot, the time will be displayed correctly again. You can also correct the time by running the program UPTIME. This gets the time from the real time clock and calls DOS to set the time.

When debugging a program that uses EMS and EEMS, you get this error message when you try to load SoftICE: "SoftICE cannot run with other 80386 control programs".

Some 386 systems come with a control program that uses the 80386 paging system to give you EMS and EEMS with a board that only has extended memory. The 386 only allows one control program at a time, so SoftICE can not coexist with these control programs. You can, however, use a true expanded memory board to debug programs that use EMS and EEMS.

SoftICE does not cause your software debugger to break.

Some software debuggers will break only when used with one type of debugging interrupt. Refer to the ACTION command in section 5.4. This lists three different types of standard action that can be taken when a break point happens. Try all three. Different ones work better for different debuggers.

SoftICE does not come up when your monitor is in graphics mode, or it does not restore your graphics screen correctly.

SoftICE does not use the ROM BIOS for its output, it must go directly to the hardware. SoftICE was designed to work with the following types of controllers, or ones that are 100% compatible: CGA MDA Hercules EGA VGA If your controller is not one of these, or not 100% compatible, you can use a second controller and monitor, and use the ALTSCR command described in section 5.9.

The key sequence used to bring up SoftICE conflicts with an existing program that you are running.

You can set a different key sequence to bring up SoftICE by using the ALTKEY command. If this doesn't work, add the SHIFT key to the current key sequence and use this new key sequence to bring up the existing program. SoftICE will not respond to the new key sequence, and will allow it to go through to the existing program. Refer to the ALTKEY command in section 5.8.

When your program crashes, SoftICE will not come up.

Refer to the BREAK command in section 5.4. This command allows you to pop up the SoftICE window when the system is hung with interrupts disabled.

After your break point triggers your debugger, your debugger does not respond.

There are two possible reasons why this problem could occur: 1) Your debugger has caused DOS or ROM BIOS to be re-entered. DOS and ROM BIOS are not fully re-entrant, so your debugger may not work correctly. Use the WARN command to turn re-entrancy warning mode on. The next time DOS or ROM BIOS is about to be re-entered, a warning message will be displayed, and you will be able choose to return to SoftICE to avoid the problem Refer to the WARN command in section 5.4. 2) A break occurred in the middle of an interrupt routine. Some debuggers can not handle this occurrence. Use ACTION set to HERE, because SoftICE will allow you to break in the middle of an interrupt routine. Refer to the ACTION command in section 5.4.

You are using a CGA monitor and you get lots of flickering when SoftICE comes up.

Certain types of video cards will flicker if characters are output without waiting for horizontal or vertical retrace. To reduce the flickering, turn FLICK mode ON. Refer to the FLICK command in section 5.9.

When you use the BOOT command, the system starts to reboot but then hangs.

SoftICE uses the interrupt 19 method of soft-booting. There are two possible times when this method could fail:

- On a freshly booted system this method will work fine. But if the system has been corrupted by an errant program, there is a chance that this method will not work.
- Some programs that use extended or expanded memory, such as EMS drivers or disk caches, are not able to handle an interrupt 19 style boot. When debugging device drivers and boot loaders that have this problem, you should use the following method. Boot the system without the drivers that cause the problem; load SoftICE; set up the drivers to load on the next boot; and then use the BOOT command. Refer to the BOOT command in section 5.3.

You just used the SYSREQ key sequence to bring up SoftICE, and your system appears to be hung, or it begins to dump the screen to your printer.

On some keyboards, you must press the ALT key and the PrtSc key simultaneously to generate a system request. If you accidentally press only the PrtSc key, the system will attempt to print your screen. If no printer is attached, your system will appear to be hung. To avoid this problem, be careful to press both keys simultaneously, or use the ALTKEY command to change to a different key sequence. Refer to the ALTKEY command in section 5.8.

You were unassembling instructions, or editing or displaying memory when your debugger crashed.

You accessed an address that triggered a SoftICE break point, and ACTION was not set to HERE. When SoftICE brings you to the point where you want to look around in memory with your debugger, you should disable the SoftICE break points. If you don't you could set off a break point unintentionally. This would cause your debugger to trigger itself, which can be a fatal problem with debuggers that cannot be re-entrant.

After you exited from your debugger, the system crashed.

This problem of course could have many causes, but one possible cause is that you may have forgotten to disable the SoftICE break points, and ACTION is still set to trigger your debugger. When the break point occurs, ACTION will attempt to trigger your debugger, but your debugger is no longer loaded.

You set a break point to trap on Interrupt 15H, function 87H, 88H, or 89H, and the break point did not occur.

SoftICE processes these functions internally in protected mode, so you cannot set break points on these functions.

Your program does not accept keystrokes, but the keyboard is still active.

A shift state key may be logically stuck down. Try pressing and releasing each shift, control and alt key.

SoftICE does not restore your graphics display properly.

SoftICE has an enhanced video virtualization mode that can virtualize many special graphics modes. Turn this mode on by entering WATCHV ON. See the description of the WATCHV command for more details. For non-compatible video controllers and certain obsure modes, you may have to use an alternate monitor. See the ALTSCR command.

The machine locks up while you are in SoftICE or moving the SoftICE window.

SoftICE has timing problems with some keyboards. Use the NOLEDS statement in S-ICE.DAT. This prevents SoftICE from sending LED commands to the keyboard.

Index

T

U

V

W