

CHAPTER 3

Debugging In 30 Minutes

- 3.1 Introduction
- 3.2 Popping Up the Window
- 3.3 Returning From the Window
- 3.4 Changing the Window Size
- 3.5 Moving the Window
- 3.6 Line Editing Keystrokes
- 3.7 Interactive Status Line
- 3.8 Command Syntax
 - 3.8.1 Specifying Memory Addresses
- 3.9 Function Keys
- 3.10 Help
- 3.11 Tutorial

15

3.1 Introduction

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

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

3.2 Popping Up the Window

You can bring up the window at any time after installing Soft-ICE. You initially bring up Soft-ICE by pressing the CTRL and D keys. However, this sequence can be changed by using the ALTKEY command (see section 5.8).

3.3 Returning From the Window

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

3.4 Changing the Window Size

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

16

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 section 5.9). 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.

3.5 Moving the Window

The Soft-ICE 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 ↑ -- moves the window one row up
- CTRL ↓ -- moves the window one row down
- CTRL → -- moves the window one column right
- CTRL ← -- moves the window one column left

3.6 Line Editing Keystrokes

Soft-ICE'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

17

following key sequences help you edit commands in the command window:

- -- moves the cursor to the right
- ← -- 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
- ↑ -- displays the previous command
- ↓ -- displays 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 -- 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.

3.7 Interactive Status Line

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

18

3.8 Command Syntax

Soft-ICE is a command-driven debugging tool. To interact with Soft-ICE, 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. All 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

3.8.1 Specifying Memory Addresses

Many Soft-ICE 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 Soft-ICE command. The public symbols must have been loaded with the Soft-ICE program loader (LDR.EXE). See chapter 7 (Symbols and Source) for a complete description of using public symbols.

The Soft-ICE 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.

3.9 Function Keys

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

The default S-ICE.DAT that comes on the Soft-ICE 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;)

21

- F10 -- Program steps (^P;)
- F11 -- Go to return address (large model)
(^G@SS:SP;)
- F12 -- Displays Soft-ICE 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 section 5.8 for a description of the FKEY command, or chapter 6 for a description of pre-initializing function keys in S-ICE.DAT.

3.10 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

22

3.11 Tutorial

The following tutorial demonstrates a few of the features Soft-ICE and gives you the opportunity to try using Soft-ICE. Soft-ICE can be used in conjunction with another debugger or as a stand-alone debugger. The tutorial demonstrates using Soft-ICE as an assistant to the DOS debugger, DEBUG, and then shows how Soft-ICE 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 Soft-ICE can be used as a stand-alone debugger.

Users who need to use Soft-ICE 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 Soft-ICE. It is recommended that you experiment with Soft-ICE 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 Soft-ICE features. Feel free to experiment on your own after going through the tutorial.

Since Soft-ICE 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 Soft-ICE from the command line. When loading Soft-ICE from the command line you can not load symbols or source files. In this case you must improvise in the last

23

section of the tutorial where Soft-ICE is used as a stand-alone debugger.

Soft-ICE 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 Soft-ICE in CONFIG.SYS with at least 50K of extended memory reserved for symbols and source files. Soft-ICE 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 Soft-ICE 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 Soft-ICE displays a copyright notice, a registration number, the name of the person who owns this copy of Soft-ICE, and the amount of extended memory reserved for each Soft-ICE component. On a system with 384K of extended memory the initial screen looks like:

```
Soft-ICE Your Name Your Company Name Registration # SInnnnnn  
Copr. (C) Nu-Mega Technologies 1987-1989 All Rights Reserved Soft-ICE Version 2.00 Soft-ICE is  
loaded from 00132000H up to 00160000H. 50K of symbol space reserved. 10K of back trace space  
reserved. 200 K of extended memory available.
```

24

The "Soft-ICE is loaded ..." message tells you the exact area of memory that Soft-ICE 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 Soft-ICE window up and give it a test drive.

Clear the screen by entering:

CLS

Bring up the Soft-ICE window by pressing:

CTRL D

The Soft-ICE window is now on the screen. If you have file S-ICE.DAT accessible from your path then the Soft-ICE 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

25

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 Soft-ICE by entering X.
2. Unload Soft-ICE 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 Soft-ICE as an assistant to another debugger.

Enter:
WIN

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

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

The Soft-ICE 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

26

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

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.

27

The Soft-ICE 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.

28

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 Soft-ICE window just disappeared.

To return to the Soft-ICE window, release the ALT key, then press:

```
ALT D
```

The window returned.

To see some previous commands, press:
the key a few times.

29

Notice that Soft-ICE 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

-- Moves the cursor one column to the right

-- 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 Soft-ICE.

The WR command is assigned to the function key F2 in the Soft-ICE 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.

30

Extend the vertical size of the Soft-ICE 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 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 Soft-ICE 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 Soft-ICE is a system level debugger that pops up wherever the

instruction pointer happens to be when you press the Soft-ICE 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.

31

Press the F12 function key.

The F12 function key defaults to be assigned to the Soft-ICE VER command. It displays the Soft-ICE copyright message and the version number.

We will now assign the F12 function key to the Soft-ICE RS command.

Enter:

RS

This will temporarily show the program screen without the Soft-ICE window.

Press the space bar to get back to get back the Soft-ICE 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 Soft-ICE window and the program screen. Now make sure the Soft-ICE 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 Soft-ICE window by entering:

CLS

32

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 Soft-ICE window.

You can also assign a sequence of Soft-ICE commands to a function key. Remember to place a carriage return between each command.

Now let's prepare to use Soft-ICE 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 Soft-ICE to generate interrupt 3's when break point conditions are met. That's how Soft-ICE will communicate with DEBUG. The default setting is HERE. ACTION HERE will cause control to return directly to Soft-ICE. Use ACTION HERE when using Soft-ICE 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 section 5.4).

33

To make the Soft-ICE window disappear again, enter:
X

This is an alternative method to exit from Soft-ICE. This especially useful in function key definitions.

Now that you are familiar with some of the basics of using Soft-ICE, 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.

34

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

Page 55,80
Title Sample program for Soft-ICE tutorial

```
DATA Segment Public 'Data'
pad    db 12H dup(O)
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
```

```
    35
```

```
    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

36

Jed has been using DEBUG but has not been able to pinpoint the problem. As a recommendation from his nephew Jethro, Jed has purchased Soft-ICE. 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 Soft-ICE 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 Soft-ICE, Jed decides to set a range break point across his code segment.

Press:
ALT D

The Soft-ICE 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 Soft-ICE to break on a write. We want to catch any unexpected writes to Jed's code.

37

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.

38

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 Soft-ICE 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
```

```
39
```

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 Soft-ICE, then exit from the debugger, by entering:

X
Q

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

To prepare Soft-ICE to debug at source level it must have been installed in your CONFIG.SYS file, and extended memory allocated for symbols and source files. Soft-ICE can only be used as a source level debugger if you have extended memory on your system. If you do not have

40

extended memory you may still want to read through the rest of the tutorial to see the capabilities of Soft-ICE 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 Soft-ICE as a stand-alone debugger we must use the Soft-ICE 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 Soft-ICE with SAMPLE.EXE loaded into memory. Notice that Soft-ICE occupies the full screen. Soft-ICE 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.

41

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

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 Soft-ICE 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 Soft-ICE back up with ALT D.

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

42

This command forces the sample program to exit. The R tells Soft-ICE to restore the interrupt vectors to the state they were when the sample program was loaded with LDR. The D tells Soft-ICE 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, Soft-ICE 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 Soft-ICE 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.

```
43
```

Press ALT D.

Press a non-space key.

We're back in Soft-ICE. 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 Soft-ICE 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 Soft-ICE interactive assembler to fix the problem.

Enter:
A not_space

Soft-ICE 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 Soft-ICE.

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 Soft-ICE to come up even when the system would normal be hung.

Exit from Soft-ICE 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 Soft-ICE 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.

46

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.

47

Blank

48

SECTION II -- Commands

Section II contains syntax listings for each Soft-ICE 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 Soft-ICE 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

49

second word is the segment offset. The addresses can be constructed of registers, expressions, and symbols.

The 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.