CHAPTER 1 - Introduction

01.01 Product Description
01.02 Using This Manual
01.03 System Requirements

01.01 Product Description

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

Soft-ICE uses 80386 protected mode to run DOS in a virtual machine. This gives Soft-ICE complete control of the DOS environment. Soft-ICE 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.

Soft-ICE was designed with three goals in mind:

* 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.).
* To work with existing debuggers. We wanted to provide a tool that worked with existing tools. We designed Soft-ICE in such a way that you don't have to learn a new debugger to get powerful hardware debugging capabilities.
* To be a user-friendly program with a window that pops up instantly and does not get in the way. All of the Soft-ICE commands were designed to fit in a small window so that information on the screen behind Soft-ICE could still be viewed. Dynamic on-line help assists users who only use Soft-ICE occasionally.

The Soft-ICE 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 Soft-ICE; even if DOS clobbered, Soft-ICE will still work
* the ability to configure Soft-ICE 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

NOTE : Soft-ICE will work with real address mode programs only. It will not work with programs that use 80286 or 80386 protected mode instructions.

01.02 Using This Manual

The Soft-ICE manual is divided into four main sections:

* Learning Soft-ICE (Section I)
* Commands (Section II)
* Support Features (Section lII)
* Advanced Topics (Section IV)

Soft-ICE can be used for most debugging problems after reading Section I, Learning Soft-ICE, and a little experimentation. Soft-ICE's user-friendly on-line help can be used to reference command descriptions and syntax.

The Learning Soft-ICE section contains installation instructions, a description of the user interface, and a tutorial. The tutorial is designed to get you up and running quickly.

The Commands section describes all of the Soft-ICE commands. The command descriptions are organized by functional group with an alphabetic index for reference.

The Support Features section covers advanced loading options, symbolic and source level debugging, and EMM 4.0 capability.

The Advanced Topics section covers topics such as using Soft-ICE with DOS loadable drivers and using Soft-ICE with non-DOS operating systems.

Throughout the manual, especially in the tutorial and the command section, examples are given that require you to give data to Soft-ICE. When the directions specify that you "press" a key, such as the key, you should press the key labelled. When the directions tell you to "enter" a phrase, such as WIN, you should type in the specified letters, then press the ENTER key.

01.03 System Requirements

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

Soft-ICE works best with extended memory, but works fine with conventional memory systems.

Soft-ICE 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. Soft-ICE also has support for a two- monitor configuration, which can be very helpful when debugging video- intensive programs.

CHAPTER 2 - Getting Started

02.01 The Diskettes
02.02 Loading Soft-ICE
02.02.01 Loading Without Extended Memory
02.02.02 Loading With Extended Memory
02.02.03 Configuring Soft-ICE for a Customized Installation
02.03 Unloading Soft-ICE
02.04 Reloading Soft-ICE

02.01 The Diskettes

Soft-ICE comes on either a 5 1/4" diskette or a 3 1/2" diskette.

When you run Soft-ICE, the name of the person that your copy of Soft-ICE is licensed to is displayed on the screen as a deterrent to software pirates. The Soft-ICE 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 Soft-ICE diskette will show the following files:

S-ICE.EXE	is the Soft-ICE program.
S-ICE.DAT	is the Soft-ICE initialization file.
LDR.EXE	is the Soft-ICE program and symbol file loader.
MSYM.EXE	is the Soft-ICE symbol file creation program.
EMMSETUP.EXE	is a program that allows you to customize the way your system uses expanded memory.
UPTIME.EXE	sets the time to that of the real time clock.
README.SI	is a text file containing information about Soft-ICE 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.

02.02 Loading Soft-ICE

Before running Soft-ICE, 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 Soft-ICE's features, S- ICE.EXE must be loaded as a device driver in CONFIG.SYS.

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

02.02.01 Loading Without Extended Memory

When no extended memory is present, Soft-ICE loads it at the highest memory location possible. The memory used by Soft-ICE is then 'mapped out', making it invisible to DOS programs. Since the total memory visible to DOS its programs is less after Soft-ICE loads, it is recommended that you load Soft-ICE before any TSR's or control programs.

If you do not have extended memory, simply enter: S-ICE

02.02.02 Loading With Extended Memory

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

When loaded as a driver, Soft-ICE 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 Soft-ICE works best if it is the first loadable device driver installed in CONFIG.SYS. For users that are new to Soft-ICE it is advisable to load Soft-ICE 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 Soft-ICE at system initialization and will be adequate for the tutorial. However, Soft-ICE will not be installed for some of its more powerful features such as EMM 4.0. You can reconfigure Soft-ICE with those features enabled after you have experimented a bit. If you already have experience with Soft-ICE or would like to set up Soft-ICE with those features immediately, please read chapter 6 (Soft-ICE 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 Soft-ICE from the DOS Prompt by typing S-ICE. Before actually loading, Soft-ICE will display a loading message and prompt. To prevent this prompt, place the word EXTENDED in the S-ICE.DAT file. See section 6.4 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, Soft-ICE occupies ZERO bytes of conventional memory. The command you use is:

S-ICE

Notes : You can NOT enable all of Soft-ICE's features when Loading from the command line. If you will be using Soft-ICE as a stand-alone debugger, it is recommended to Load Soft-ICE from CONFIG.SYS.

If you want to load Soft-ICE as a device driver, but don't want Soft-ICE to be resident all of the time, you should use the /UN loading switch. Refer to section 6.3.1 for more information.

02.02.03 Configuring Soft-ICE for a Customized Installation

You can customize Soft-ICE with Soft-ICE loading switches in CONFIG.SYS and with the Soft-ICE initialization file S-ICE.DAT. The CONFIG.SYS loading switches allow you to customize how the extended memory will be reserved by Soft-ICE. 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 Soft-ICE. for more information about customizing Soft-ICE, refer to chapter 6.

02.03 Unloading Soft-ICE

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

S-ICE /U

This command places the machine back in real address mode. If Soft-ICE was initially loaded from CONFIG.SYS When the memory is still reserved for Soft-ICE and can not be used by other software. If Soft-ICE 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 Soft-ICE could crash your system.

02.04 Reloading Soft-ICE

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

S-ICE

```
                    DOS Debugging for 68 Machines

                    Contents

                    01 Preparing to the debugger
                    02 Entering the debugger
                    03 Displaying memory contents
                    04 Displaying the CPU registers
                    05 Displaying the disassembly
                    06 Displaying the Stack Frame
                    07 Displaying the Stack
                    08 Line Editing Keystrokes
                    09 Line Editing Commands
                    10 Line Editing Keystrokes
                    11 Line Editing Commands
                    12 Memory Dump Commands
                    13 Function Keys
                    14 Exit Keys
                    15 Quit Keys

                    01 Preparing to the debugger

   Is it unfortunate world DOS 1.0 Dragon Dos phase through a window that not be expected as a result of DOS 1.0 Dragon Dos phase through a window that not be expected as a result of DOS 1.0 Dragon Dos phase through a window that not be expected as a result of
```

```
04.00 - Using Break Point Commands

  04.00 Notational Conventions
  04.01 Introduction
  04.02 Setting Break Points
  04.03 Deleting Break Points

04.00 Notational Conventions

  Syntax descriptions use various listings for each DOT-II command, and
  explanations and examples for each command. All numbers, when
  hexadecimal, any number can be an expression using '+,-,*,/', or
  parentheses. Any number can be a hexadecimal digit, but any number
  starting with a decimal digit (0-9) must be preceded 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.
  ...       Ellipsis
  | or ,    Single brackets enclose a list of items or choices.

  ;         Vertical bars separate alternatives. Use item a or item b.

  COMMA     Used in a syntax value that specifies the number of item break
            point conditions must be met before the actual break point
            occurs. Value can be expressed as a decimal, hex, or a value
            item that DOT-II WATCH is brought up, the value item to reach
            the value originally specified.

  TYPE      Used in a syntax value that specifies what type access the break
            point will apply to. Valid types are 'R' for reads, 'W' for write
            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 offset address. Valid values are numbers, register
            registers, variables, and symbols. The address may also contain
            the special characters '$', '.', and '@'. See section 3.0
            Expression Syntax for a description of these special characters.

  BYTE      Break number is an identification number that identifies the
            break point. Used in a syntax value that specifies deleting,
            editing, editing, enabling, or disabling items. The break number
            can be a hexadecimal digit from 0 to F.

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

  mode      Mode is a bitmask that is represented as combination of 1's,
            0's, and X's. X's are don't-care bits.
  07, 17 and 27
            Command specifiers that unsigned comparisons of values.

  Example : BP20 21 x 4Q0 W 1XXX XXXX

This command causes a break point to occur if port 20H is written
to with the high order bit set.

04.01 Introduction

DOT-II2 has break point capability that has traditionally only been
available in hardware debugging tools. Hardware debugging. DOT-II2
utilizes 386 advanced break point capability instead of more additional
hardware.

Break points can be set on memory location reads and writes, memory
range reads and writes, program execution and port accesses. DOT-II2
allows up to 255 break points to be set at any time. Each break point
break point number is used to identify break points when you set delete,
disable, enable, or edit them.

Break point conditions are very sticky. That means they don't
change automatically after they've been used, you must
intentionally clear or disable them using the BC or the BD commands.
Break point conditions can be set to cause a break point to occur
after break points of a single type except for break points on memory
location (BPMW), 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
specifies how many times the break point condition must be
satisfied before the break point action occurs.

04.02 Setting Break Points

BPM, BPMB, BPMW, BPMD set break point on memory access or execution
BPM set break point on memory range
BPMB set break point on byte access
BPMW set break point on word access
BPMD set/clear break point on execution
BPIO set break point on port access
BPIOW set break point on word access
BPMWD Mode for execution break points is error

Set break point on memory access or execution

  Syntax :

                BPM[size]address[verb][qualifier value][C<count]

  Size :
                Byte, Word, DiskSwap(D)
                The size is optional a range covered by this break point.
                For example, if double word is used, and the third byte of
                the double word is accessed, the break point will occur. The
                size is also important if the optional qualifier is
                specified (see below).

                B, W, PW, or D
  Qualifier :
                Mask, Value, NE (Not Equal), GT(Greater than), LT (less Than),
                A (Mask)
                These qualifiers are only applicable to the read and write
                break points.
  Verb :
                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 execute.

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

                All of the verb types except A cause the program to execute the
                instruction that caused the break point to occur. DOT-II2 will
                be the instruction after the break point. If the verb type is A,
                the current CS:IP will be the instruction after the break point.

  Example : BPM B:0500 W 0 0XXX XXXX XXXX XXXX

  This command defines a byte break point on memory write. The
  break point action will occur the first time that location 0B:F00
  has a value written to it that sets the high order bit to 0 and
  the low order bit to 1. The other bits are any value.

  BPM B:1000 W 07 5

  This command defines a byte break point on memory write. The
  break point action will occur the first time that location
  0B:1000 has a value written to it that is greater than 5.

Set break point on memory range

  Syntax :

                BPM start-address-and-address [verb] [C<count]

                start-address, end-address :
                start address and end-address specify memory range.
  Verb :
                R, W, Rw, Or Rw

  Comments :

                The BPM command allow you to set a break point across a range of
                memory.

                All of the verb types except Y or Rw 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 specified with the execute verb, the execute break
                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 AK page that contains
                the range break point will cause a page fault. This performance
                degradation is usually not noticeable. However, degradation could
                be extreme in exception cases.

                The Y and Rw verbs enable back trace ranges on the specified
                range. They do not cause break points, but instead log
                information about the instructions executed in the range in the DmW
                or TMEM commands. For more information on back trace ranges, see
                chapter 9.

  Example :

  BPM 0000:0 0000:1000 W

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

Set break point on I/O port access

  Syntax :

                BPIO port [verb] [qualifier value] [C<count]

  Port :
                A byte or word value.
  Verb :
                R (IN), W (OUT), or RW
  Qualifier :
                Mask, Value, NE (Not Equal), GT (Greater than),
                A (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 to the port. If a mask is specified with the
                port, the value may be a byte or a word. If the I/O is a byte
                operation, only the low byte of the value will be compared.

                The instruction pointed to by the current CS:IP is 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 FF

  This command defines a break point on I/O port access. The break
  point will occur the first time that a byte operation is made to
  port 21h with a value other than FFH.

  BPIO 200 R 1234 XXXX

  This command defines a break point on I/O port access. The break
  point will occur the first time that a word operation is made to
  port 200h with a value of 1234h.

  The instruction pointed to by the current CS:IP is 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 FF

  This command defines a break point on I/O port access. The break
  point will occur the first time that a byte operation is made to
  port 21h with a value other than FFH.

  BPIO 200 R 0 1XX XXXX

  This command defines a break point on I/O port read. The
  break point will occur the first time that a word operation is
  made to port 200h with a value that has the high order bits set to 1. The
  other bits can be any value.

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 specifying 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 (AX,
                AH, or AL). If the comparison is true, the break point occurs the
                specified register, then the break point will occur.

                When the break point occurs, if the interrupt was a hardware
                interrupt, the instruction pointed CS:IP:ip will point to the
                first instruction within the interrupt routine. The INT0 command
                will be the instruction after the interrupt that caused the
                occurred. If the interrupt was a software interrupt, when break
                point occurs occurs, the instruction pointed CS:IP:ip will point to
                the 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.

Set/Clear break point on execution

  Syntax :

                BPM [address] [C<count]

  Comments :

                The BPM 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 of the current cursor location is used to
                set or the address of the current cursor location. If an
                execution break point has already been set at the address of the
                current cursor location, the break point will be 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 :

  BPM uses an interrupt 3 style of break point unless the specified
  address is in a ROM. Since ROM cannot be written to, interrupt 3
  make more execution break points available. If circumstances
  require the use of a break point register for some reason (code
  in ROM, for example) you can set an execution break point
  with the BPM command.

  Example :

  BPM 1234

  This sets an execution break point at source line 1234.

Set CS:IP range qualifier

  Syntax :

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

  Off :
                When NOT is specified, the break point will occur when
                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
                has been corrupted or accidentally modifying code outside of its
                boundaries.

                When break point conditions are met, the CS:IP registers are
                checked against the specified range. If the CS:IP is within 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.

Wait for multiple break points to occur

  Syntax :

                BPWAD [OR] | [" [AND]
  List :
                A series of break-numbers separated by commas or spaces
  Comments :

                AND#s Together all break points

  Comments :

                The BPWAD 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 BPWAD
                command allows you to specify a number of break points to
                occur before the action is generated. This function allows more
                complex break point conditions to be set.

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

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

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

  Example :

  BPWAD [OR] | [" [AND]
  List :
                A series of break-numbers separated by commas or spaces
  Comments :

                AND#s Together all break points

  Comments :

                The BPWAD 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 BPWAD
                command allows you to specify a number of break points to
                occur before the action is generated. This function allows more
                complex break point conditions to be set.

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

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

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

04.03 Manipulating Break Points

DOT-II2 provides several commands for manipulating break points.
Break points can be deleted, edited, enabled, disabled, listed,
and disabled all 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 Set break point as a template
BC Clear break points

Disable break points

  Syntax :

                BD [,]

                This command temporarily disables break points 1 and 3.

Enable break points

  Syntax :

                BE list [,*]

  List :
                A series 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 point) command.

                Note that a break point is automatically enabled when defined.

  Example :

                BE 3 This command enables break point 3.

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 the break point is ORed with the previous
                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:

                B1 BPMW 1234:0000 W 2Q 0010 C=01
                B2 BPMW 1234:0000 W 2Q 0010 C=01
                B3 BPIO 0021 W 4W 0007 C=01
                B4 BPIOW 21 AH=4C C=01

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

Edit break point

  Syntax :

                BPE 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
                cursor keys, and entered by pressing the ENTER. This command
                offers a quick way to modify the parameters of an existing break
                point.

  Example :

                BPE 5 This command moves a description of break point 5 into the edit
                line. When the ENTER key is pressed, a new break point is added.

  Example :

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

Clear break points

  Syntax :

                BC list | *

  List :
                A series of break-numbers separated by commas or spaces

                AND#s Together 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.


```

```
NAME
       ftw, nftw - file tree walk
SYNOPSIS
       #include <ftw.h>
       int ftw(const char *, int (*)(const char *,
                   const struct stat *, int), int);
       int nftw(const char *, int (*)(const char *,
                   const struct stat *, int, struct FTW *),
                   int, int);
DESCRIPTION
       ftw() recursively traverses a directory hierarchy
       rooted in path.  For each file in this hierarchy
       ftw() calls the function func() with the path
       of the file, a pointer to a struct stat
       containing information about the file, and an
       integer giving additional information.  Unless
       ftw() encounters a file that it does not have
       permission to stat(), or a directory that it
       does not have permission to read, or long path
       names, func() is called for every file in the
       hierarchy.  When a directory is encountered,
       ftw() first calls func() with the path of the
       directory, then lists the files in the
       directory, and then traverses the directories
       and files in the directory in the same manner.
       ftw() uses at most one file descriptor in
       order to do its work.  The tree traversal
       continues until a file is encountered for
       which func() returns a nonzero value, or until
       all files in the tree have been processed.
       ftw() returns the value returned by func().
       If func() returns 0, then ftw() will continue
       traversing the tree.  If func() returns 1,
       then ftw() will stop traversing the tree and
       return 1 to its caller.  If func() returns 2,
       then ftw() will not descend into the directory
       that was passed to func().  If func() returns
       -1, then ftw() will stop traversing the tree
       and return -1 to its caller.  If func() returns
       a value other than 0, 1, 2, or -1, then the
       behavior of ftw() is unspecified.  If func()
       returns 2 when a directory is encountered,
       then ftw() will not descend into the
       directory, but will continue to call func()
       for other files and directories in the tree.
       nftw() is similar to ftw() except that func()
       is called with an additional fourth argument.
       The fourth argument to func() is a pointer to
       a struct FTW.  The contents of this structure
       are implementation-defined.  The fourth
       argument to func() may be used to obtain
       additional information about the traversal
       state.  The nftw() function takes an additional
       argument flags which is zero, or the bitwise
       OR of one or more of the following flags:
       FTW_CHDIR
              When set, nftw() changes the current
              working directory to each directory as it
              reads through the directory.  If set,
              pathnames relative to the current
              directory may be used.  If not set, the
              current working directory is not changed.
       FTW_DEPTH
              When set, nftw() lists the files in a
              directory before the directory itself.
              If not set, the directory is listed
              before its files.
       FTW_MOUNT
              When set, nftw() does not descend into
              directories that are mount points.
       FTW_PHYS
              When set, nftw() does not follow symbolic
              links.  If not set, symbolic links are
              followed.
       FTW_SEARCH
              When set, nftw() does not descend into
              directories that are not searchable.
       If func() returns 2 when a directory is
       encountered, then nftw() will not descend into
       the directory that was passed to func().
       If func() returns 2 when a directory is
       encountered, and the FTW_DEPTH flag is set,
       then nftw() will not descend into the
       directory that was passed to func().
       If func() returns 2 when a directory is
       encountered, and the FTW_SEARCH flag is set,
       then nftw() will not descend into the
       directory that was passed to func().
       If func() returns 2 when a directory is
       encountered, and the FTW_MOUNT flag is set,
       then nftw() will not descend into the
       directory that was passed to func().
       If func() returns 2 when a directory is
       encountered, and the FTW_PHYS flag is set,
       then nftw() will not descend into the
       directory that was passed to func().
       If func() returns 2 when a directory is
       encountered, and the FTW_CHDIR flag is set,
       then nftw() will not change the current
       working directory to the directory that was
       passed to func().
       In a multithreaded process, ftw() and nftw()
       behave as if they do not change the process
       current working directory.
       ftw() and nftw() are not thread-safe.
RETURN VALUE
       ftw() and nftw() return 0 if the entire
       directory tree was successfully traversed.
       If func() returns a nonzero value, then ftw()
       and nftw() stop traversing the tree and return
       the value returned by func().  If a directory
       cannot be read, or if memory for data
       structures cannot be allocated, then ftw() and
       nftw() return -1 and set errno to indicate the
       error.  If a file cannot be stat(), then ftw()
       and nftw() do not call func() for that file.
       If a directory cannot be read, then ftw() and
       nftw() do not descend into that directory.
       If nftw() is called with an invalid value in
       flags, then nftw() may fail with the error
       EINVAL.
ERRORS
       ftw() and nftw() may fail if:
       [EACCES]
              Search permission is denied for any
              component of path, or read permission is
              denied for a directory that is being
              processed.
       [ELOOP]
              More than MAXSYMLINKS symbolic links were
              encountered while traversing the pathname
              in path.
       [ENAMETOOLONG]
              The length of path exceeds PATH_MAX and
              {_POSIX_NO_TRUNC} is in effect.
       [ENOENT]
              A component of path does not name an
              existing file or path is an empty string.
       [ENOTDIR]
              A component of path is not a directory.
       [EOVERFLOW]
              Some of the information cannot be
              represented in a data structure.
       nftw() may fail if:
       [EINVAL]
              The value of flags is invalid.
       [EMFILE]
              Too many file descriptors are currently in
              use by this process.
       [ENFILE]
              Too many files are currently open in the
              system.
       [ENOMEM]
              Insufficient storage space is available.
       [ENOTSUP]
              The implementation does not support the
              combination of flags to the function.
       [EOVERFLOW]
              Some of the information cannot be
              represented in a data structure.
EXAMPLES
       The following is a sample binary utility to display
       directory trees:
       #!/usr/bin/awk -f
       BEGIN {
              exit ftw(ARGV[1], "myftw", 1)
              print "ftw() failed" >> "/dev/stderr"
              exit 1
       }
       function myftw(path, stat, type, ftw) {
              if (type == "d")
                     printf "%s/ %d\n", path, ftw[2]
              else if (type == "dnr")
                     printf "%s/ %d (unreadable)\n",
                     path, ftw[2]
              else if (type == "dp")
                     printf "%s/ %d (postorder)\n",
                     path, ftw[2]
              else if (type == "slnr")
                     printf "%s -> ? (unreadable)\n",
                     path
              else if (type == "sl")
                     printf "%s -> ?\n", path
              else if (type == "sln")
                     printf "%s -> ? (nonexistent)\n",
                     path
              else if (type == "f")
                     printf "%s %d\n", path, ftw[2]
              else if (type == "noinfo")
                     printf "%s ?\n", path
              else if (type == "ns")
                     printf "%s ? (unstatable)\n",
                     path
              else if (type == "err")
                     printf "%s ? (error)\n", path
              else
                     printf "%s ? (%s)\n", path, type
              return 0
       }
APPLICATION USAGE
       The ftw() and nftw() functions operate on the
       pathnames of files and directories. If the
       application modifies the pathnames in the
       directory tree, or creates new files in the tree
       while these functions are traversing the tree,
       the results are unspecified.
       Since ftw() and nftw() are not thread-safe, the
       application should use the thread-safe
       alternatives pthread_ftw() and pthread_nftw()
       when it is operating in a multiple-thread
       environment.
RATIONALE
       The ftw() function is an XSI extension.
       The nftw() function is an extension derived from
       the Solaris 2.6 and 7 functions of the same
       name.
       The ftw() and nftw() functions are not thread-
       safe because they do not provide a mechanism to
       ensure that the current working directory
       remains unchanged while the function is
       traversing the tree. The functions may
       alternately change the current working
       directory and return to the original directory
       while calling the user-specified function, and
       if the application has other threads that
       depend on the current working directory, they
       will be affected. Since the functions cannot
       restore the current working directory, they
       also cannot be used to access files deeper than
       PATH_MAX bytes in length. The pthread_ftw() and
       pthread_nftw() functions are thread-safe and
       do not change the current working directory.
       The nftw() function provides a superset of the
       ftw() function. The additional arguments to
       func() and the flags to nftw() provide
       additional information and more control over
       the traversal. For example, nftw() reports
       directories after their contents when the
       FTW_DEPTH flag is set, and it stops descending
       into a directory if func() returns 2 when it
       is called with a directory. The ftw() function
       may be implemented in terms of nftw().
       The ftw() function uses the obsolescent interface
       to stat(). The nftw() function uses the
       preferred interface.
       The ftw() and nftw() functions are not required
       to detect all cycles in the file system when
       following symbolic links. Such cycles can
       occur if the application or another process
       creates a hard link to a directory in the file
       system, or if the file system is not a strict
       hierarchy. The ftw() and nftw() functions are
       required to detect such cycles if they are
       caused by symbolic links.
FUTURE DIRECTIONS
       None.
SEE ALSO
       chdir(), fstatat(), lstat(), stat()
       The Base Definitions volume of IEEE Std 1003.1-
       2024, <ftw.h>
CHANGE HISTORY
       First released in Issue 3.
       Entry added for thread-safety.
       Entry added for Large File Summit extension.
       The prototype for ftw() changes from int (*)(char
       *, int (*)(char *, struct stat *, int), int);
       to int (*)(const char *, int (*)(const char *,
       const struct stat *, int), int);.
       The prototype for nftw() changes from int
       (*)(char *, int (*)(char *, struct stat *, int,
       struct FTW *), int, int);
       to int (*)(const char *, int (*)(const char *,
       const struct stat *, int, struct FTW *), int,
       int);.
       The prototypes for ftw() and nftw() changes
       from int (*)(char *, int (*)(char *, struct
       stat *, int), int);
       to int (*)(const char *, int (*)(const char *,
       const struct stat *, int), int);.
       The prototypes for ftw() and nftw() changes
       from int (*)(char *, int (*)(char *, struct
       stat *, int,
```

- Load Stores
- Compare
- Compare carried bank and stores carry
- Compare carried bank and stores
- Operate carried bank and stores

Compare

The compare instruction sets the carry flag to reflect the operation of the specified register minus the specified memory location. The carry flag is set if the register is greater than or equal to the memory location, and cleared if the register is less than the memory location. It is not affected by the zero flag.

Compare carried bank and stores carry

Sets carry and stores carry flag to reflect the operation of the specified register minus the specified memory location. The carry flag is set if the register is greater than or equal to the memory location, and cleared if the register is less than the memory location. It is not affected by the zero flag.

Operate carried bank and stores

Sets carry and stores carry flag to reflect the operation of the specified register minus the specified memory location. The carry flag is set if the register is greater than or equal to the memory location, and cleared if the register is less than the memory location. It is not affected by the zero flag.

Bit

Sets the zero flag according to the specified bit of the specified register. The zero flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the carry flag.

Set

Sets the zero flag according to the specified bit of the specified register. The zero flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the carry flag.

Reset

Sets the zero flag according to the specified bit of the specified register. The zero flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the carry flag.

Toggle

Sets the zero flag according to the specified bit of the specified register. The zero flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the carry flag.

Test

Sets the zero flag according to the specified bit of the specified register. The zero flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the carry flag.

Test and set

Sets the zero flag according to the specified bit of the specified register. The zero flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the carry flag.

Test and reset

Sets the zero flag according to the specified bit of the specified register. The zero flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the carry flag.

Test and toggle

Sets the zero flag according to the specified bit of the specified register. The zero flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the carry flag.

Test and set carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test and reset carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test and toggle carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Set carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Reset carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Toggle carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test carry and set

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test carry and reset

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test carry and toggle

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test and set carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test and reset carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test and toggle carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test carry and set

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test carry and reset

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test carry and toggle

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test and set carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test and reset carry

Sets the carry flag according to the specified bit of the specified register. The carry flag is set if the specified bit is zero, and cleared if the specified bit is one. It is not affected by the zero flag.

Test and toggle carry

CHAPTER 6 - Soft-ICE Initialization Options

06.01 Introduction
06.02 Loading Soft-ICE from the DOS Prompt
06.03 Loading Soft-ICE as a Loadable Device Driver
06.03.01 Soft-ICE Loading Switches
06.04 The Soft-ICE Initialization File S-ICE.DAT
06.04.01 Special Configuration Options
06.04.02 Function Key Assignments
06.04.03 Initialization Command Sequence

06.01 Introduction

The Soft-ICE 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 Soft-ICE, it must be initially loaded as a device driver in CONFIG.SYS. However, there may be circumstances when you might want to run Soft-ICE from the DOS prompt or a batch file, such as:

* You do not have extended memory in your system Soft-ICE 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, Soft-ICE occupies approximately 2K of conventional memory.

* You only need to use Soft-ICE occasionally and there are no other programs using extended memory. In some cases you may need some of the features that require Soft-ICE to be loaded in CONFIG.SYS but do not want Soft-ICE to be resident all of the time. In this case Soft-ICE can be loaded in CONFIG.SYS to reserve extended memory, and then disabled, by using the /UN switch, until Soft-ICE is required. See section 6.3.1 for more information about the /UN switch.

06.02 Loading Soft-ICE from the DOS Prompt

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

To load Soft-ICE from the DOS prompt type:

```
S-ICE
```

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

In systems with extended memory, you should only load Soft-ICE from the DOS prompt if you are not using extended memory for anything else (e.g., VDISK, CACHE, HIMEM,...). When you initially load Soft-ICE from the command line or from a batch file, Soft-ICE will prompt you with a warning message. This warning message is just to remind you that Soft-ICE will overwrite the highest portion of extended memory when it loads. You can suppress this warning prompt with the EXTENDED option in the Soft-ICE configuration file S-ICE.DAT. For more information about the EXTENDED option, see section 6.4.1.

06.03 Loading Soft-ICE as a Loadable Device Driver

In order to use all of the Soft-ICE features, you must first load Soft-ICE 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 Soft-ICE to manage extended memory so you can run Soft-ICE with programs that use extended memory, such VDISK, CACHE and HIMEM.

* Symbolic and source level debugging Loading as a device driver allows Soft-ICE to allocate an extended memory buffer for symbols and source information.

* Back trace ranges and the SNAP command Loading as a device driver allows Soft-ICE to allocate an extended memory buffer for a back trace buffer. This buffer is also used for the Soft-ICE SNAP command.

* Enabling Soft-ICE's EMM 4.0 capability

* Running Soft-ICE with MagicCV or MagicCVW

Note : When loaded as a device driver in CONFIG.SYS, Soft-ICE 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 Soft-ICE works best if it is the first loadable device driver installed in CONFIG.SYS.

06.03.01 Soft-ICE 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 Soft-ICE. 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 Soft-ICE 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 Soft-ICE 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 7.

* /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 9.

* /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 10 for more information about running Soft-ICE 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 8 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 Soft-ICE, and you must execute S-ICE.EXE from the DOS prompt when you are ready to use Soft-ICE.

Soft-ICE 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 Soft-ICE, 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 Soft-ICE will load into the highest portion of extended memory, leaving the remaining memory starting at 100000H (one megabyte mark).

06.04 The Soft-ICE Initialization File S-ICE.DAT

Soft-ICE 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 Soft-ICE 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 Soft-ICE 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 Soft-ICE 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

06.04.01 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 Soft-ICE to use this memory. Note that the COMPAQ option is the same as the /C command line parameter in Soft-ICE 1.X.

* NOLEDS: The NOLEDS option tells Soft-ICE not to set and clear the keyboard LEDs while the Soft-ICE window is up. On some keyboards the are timing problems that will cause Soft-ICE to lose synchronization with the keyboard. If Soft-ICE hangs when you are in the Soft-ICE window use this option. Note that the NOLEDS option is the same as the /L command line parameter in Soft-ICE 1.X.

* NOTVGA: The NOTVGA option allows Soft-ICE 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 Soft-ICE 1.X.

* EXTENDED: The EXTENDED option causes Soft-ICE to load directly into extended memory without prompting the user with a warning message. It should be used if you are loading Soft-ICE 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 Soft-ICE 1.X.

06.04.02 Function Key Assignments

One or more Soft-ICE commands can be assigned to any function key at load time. See the description of the FKEY command in section 5.8 (Debugger Customization Commands) for a description of assigning function keys from the Soft-ICE 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 Soft-ICE 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 Soft-ICE dump command to function key 12. When
F12 is pressed Soft-ICE will dump at offset 100H in the current data
segment. The semi-colon following the 100 represents the ENTER key.
```

06.04.03 Initialization Command Sequence

A sequence of commands can be automatically executed when Soft-ICE loads. This is useful for customizing Soft-ICE 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 cmd 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 Soft-ICE 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.
```

CHAPTER 7 - Symbolic and Source Level Debugging

07.01 Introduction
07.02 Preparing for Symbolic or Source Debugging
07.02.01 Preparing for Symbolic Debugging Only
07.02.02 Preparing for Symbolic and Source Level Debugging
07.03 Reserving Memory for Symbols and Source Files
07.04 Loading Programs and Symbol Files
07.04.01 Loading Program, Symbols and Source
07.04.02 Loading Only Symbols and Source Files
07.04.03 Loading a Program With No Symbols or Source
07.05 Debugging With Symbols
07.06 Debugging With Source
07.06.01 Using Line Numbers
07.06.02 Using Source Mode in the Code Window

07.01 Introduction

Soft-ICE 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 Soft-ICE to provide complete system debugging in user level. You can debug TASR's interrupt routines and other systems level code at the source level.

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

07.02 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 Soft-ICE 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).

07.02.01 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 Soft-ICE 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.

07.02.02 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 either use 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.

07.03 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 Soft-ICE and reserving space for symbols and source files is:

DEVICE = S-ICE.EXE /SYM 1024

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

07.04 Loading Programs and Symbol Files

The Soft-ICE utility LDR.EXE is used for loading programs, symbol files and source files. For symbolically debugging application programs and TASR 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

07.04.01 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 Soft-ICE 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.

07.04.02 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 Soft-ICE 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

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

07.05 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 Soft-ICE 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:

1. Loading symbols for a loadable device driver

2. Loading symbols for a TASR that has already been loaded

3. Your program moves itself to a location other than it original location.

See section 5. 10 for a complete description of these commands.

07.06 Debugging With Source

When source files are loaded, Soft-ICE allows you to view and step through your source code as you are debugging. Soft-ICE 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.

07.06.01 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

07.06.02 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 Soft-ICE 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 WC command.

Toggle between source, mixed, and code modes with the SRC command.

Place a source file in the code window 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 window 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) with the EC command.

Scroll the source with the keys up, down, PageUp, 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

CHAPTER 8 - Expanded Memory Support

08.01 Introduction
08.02 Configuring the EMM Environment
08.02.01 Default EMM Pages
08.02.02 Customizing the EMM Page Map
08.02.02.01 Including and Excluding Areas from EMM
08.03 Other EMM Features
08.03.01 Increasing Conventional Memory
08.03.02 Automatic Page Frame Locating
08.04 EMM Debugging

08.01 Introduction

Soft-ICE has an expanded memory manager built into its kernel. The Soft- ICE expanded memory manager supports the Lotus-Intel-Microsoft 4.0 specification. This Soft-ICE 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 Soft-ICE. If you are using those programs for EMM capability or backfilling, you can use the Soft-ICE EMM manager in their place.

Enabling EMM capability in Soft-ICE 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 section 6.3 (Loading Soft-ICE as a Loadable Device Driver) for details of installing Soft-ICE in CONFIG.SYS.

3. Reboot your system.

08.02 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.

08.02.01 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.

08.02.02 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 configurethe EMM map enter :

EMMSETUP

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.

08.02.02.01 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.

08.03 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.

08.03.01 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 Soft-ICE. Doing so will cause your system to crash.

08.03.02 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.

08.04 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. See section 5.6 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

CHAPTER 9 - Back Trace Ranges

09.01 Introduction
09.02 Using Back Trace Ranges
09.03 Special Notes

09.01 Introduction

Soft-ICE 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.

Soft-ICE 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.

09.02 Using Back Trace Ranges

To use back trace ranges you must do the following:

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 Soft-ICE with the X command.

5. After a break point has occurred, or you have popped Soft-ICE 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.

09.03 Special Notes

While in trace simulation mode, most Soft-ICE commands work as normal, including displaying the memory map, and displaying and editing data. The exceptions are:

1. 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.

2. Commands that normally exit from Soft-ICE 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. Soft-ICE 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

CHAPTER 10 - Using Soft-ICE with MagicCV or MagicCVW

10.01 Introduction
10.02 Running Soft-ICE with MagicCV or MagicCVW
10.03 Special Considerations
10.04 The Soft-ICE ACTION command

10.01 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 Soft-ICE in combination with MagicCV or MagicCVW allows you to have the power of Soft-ICE 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.

10.02 Running Soft-ICE with MagicCV or MagicCVW

To use Soft-ICE 2.0 and MCV together, you must install S-ICE.EXE as a loadable device driver. S-ICE.EXE comes on the Soft-ICE 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 Soft-ICE. Refer to chapter 6 for information about these switches.

To run MagicCV or MagicCVW after Soft-ICE has been loaded, refer to your MagicCV or MagicCVW manual.

Notes : MagicCVW requires Soft-ICE version 2.00 or greater. MagicCV requires Soft-ICE version 1.02 or greater. The S-ICE.SYS and NUMEGA.SYS drivers were shipped with some versions of Soft-ICE. The S-ICE and NUMEGA drivers must be replaced by S-ICE.EXE before you can run MagicCV and Soft- ICE 2.0 together.

10.03 Special Considerations

Two Virtual Machines : When you are using both Soft-ICE and MCV together, you must keep in mind that CV is in a separate virtual machine from the target environment. You can pop Soft-ICE up from either virtual machine, i.e., when CV is running, or when the target program is running.

If you pop Soft-ICE up while the target program is running everything works as defined in the Soft-ICE manual. If you pop Soft-ICE 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 Soft-ICE 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 Soft-ICE 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 Soft-ICE 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 Soft-ICE 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 Soft-ICE window using the Soft-ICE X command, then do one or more CV traces to step through the target program.
* Use Soft-ICE to go to the target program address, then use the Soft-ICE T or P commands to step through your target program. To do this, exit the Soft-ICE window with the Soft-ICE 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 Soft-ICE. If the Soft-ICE window is not already in narrow mode, use the Soft-ICE WIN command to change the window size. Move the Soft-ICE window so you can see the instruction addresses on the left side of the screen. Now you can use the Soft-ICE 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 Soft-ICE 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 :Soft-ICE 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 Soft-ICE. If you do use the /R switch, Soft-ICE 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 Soft-ICE break points. The CV /R switch works when you are running MCV without Soft-ICE.

10.04 The Soft-ICE ACTION Command

The ACTION command allows three different methods activating CV from a Soft-ICE 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.

CHAPTER 11 - Advanced Features

11.01 Using Soft-ICE with other Debuggers
11.01.01 Debuggers that Use DOS
11.01.02 ACTION Command with other Debuggers
11.01.03 Special Considerations
11.01.04 Using Soft-ICE with CODEVIEW
11.01.05 Debuggers that Use 80386 Break Point Registers
11.02 User-Qualified Break Points
11.02.01 Example of a User-Qualified Break Point
11.03 The Window in Graphics Mode
11.04 Expanded Memory Debugging Features
11.05 Extended Memory Debugging Features

11.01 Using Soft-ICE with other Debuggers

Soft-ICE 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.

11.01.01 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 Soft-ICE (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.

Soft-ICE provides optional re-entrancy warning, which is activated with the WARN command. When WARN mode is on, Soft-ICE 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, Soft-ICE displays a warning message and offers you the choice of continuing to execute the code or returning to Soft-ICE.

Note that Soft-ICE itself does not use DOS or ROM BIOS calls in its debugging commands. This means that you can use Soft-ICE any time, without the worry of re-entrancy problems.

For more information on the WARN command, see section 5.4.

11.01.02 ACTION Command with other Debuggers

Different debuggers use different methods of activation For a description of these methods see section 13.1.

If you want to return to your debugger after a break point reached, you must change the ACTION (see section 5.4) to 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.

11.01.03 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, Soft-ICE 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 Soft-ICE break points once Soft-ICE has helped you get to the point where you want to look around with your debugger.

11.01.04 Using Soft-ICE with CODEVIEW

Soft-ICE 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 Soft-ICE to wake up CODEVIEW.

11.01.05 Debuggers that Use 80386 Break Point Registers

The 80386 has 4 break point registers that are available for use by debuggers. Soft-ICE uses these for its memory byte, word and double word break points. If the debugger you are using Soft-ICE 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 Soft-ICE 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 the following:

* Bring up the Soft-ICE window before you start the other debugger.
* Turn on Soft-ICE'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 Soft-ICE window and turn the Soft-ICE break mode off if desired.

11.02 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.

Soft-ICE contains a very general mechanism for calling user-written break point qualification routines: the ACTION command. When you use the ACTION command, Soft-ICE 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 Soft-ICE 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 Soft-ICE The command 13HERE must be turned on in order for the INT 3 to bring up Soft-ICE (see section 5.4).

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 Soft-ICE, 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 Soft-ICE.

3. In Soft-ICE, set ACTION to the interrupt- number that was used to point to your qualification routine.

4. In Soft-ICE, set 13HERE on. This is necessary to bring up Soft-ICE after the conditions have been met.

5. Set the Soft-ICE general break point conditions. When any of these break point conditions are met, your qualification routine will be called.

11.02.01 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 Soft-ICE :

```
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 Soft-ICE will call your break point qualification routine on every break point.

```
ACTION 99
```

Set I3HERE on so the qualification routine can activate Soft-ICE when the conditions occur.

```
I3HERE ON
```

Now you need to set the break points. For this example, we are just interested when the registers are: AX = 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 Soft-ICE is popped up.

When Soft-ICE pops up, the instruction pointer will be pointing at the INT3 in your qualification routine (9000: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 Soft-ICE 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.

11.03 The Window in Graphics Mode

The screen is switched to text mode when Soft-ICE 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 Soft-ICE window to display on a second monitor (see the ALTSCR command, section 5.9). The second feature allows you to restore the screen while you are doing P or T instruction step commands (see the FLASH command, section 5.9). The third feature allows you to restore the program screen temporarily (see the RS command, section 5.9).

If Soft-ICE does not seem to be following your program into graphics mode, try turning WATCHV on (see section 5.9 for details).

11.04 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 section 5.6 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# Page# 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# Page# 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
```

11.05 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 Soft-ICE 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
```

Using Keepalives

Keepalive Implementation Guidelines
Type of Keepalives Supported by SCTP
Heartbeat Chunk Type
Heartbeat Chunk Parameters
Heartbeat Acknowledgment Chunk Type
Heartbeat Acknowledgment Chunk Parameters
Heartbeat Acknowledgment Chunk Error Indication
Heartbeat Acknowledgment Chunk Error Indication Parameters
Heartbeat Implementation
Heartbeat Implementation Guidelines
Heartbeat Implementation Example
Heartbeat Implementation Example Conclusion
Heartbeat Implementation Example Conclusion Continued

Keepalives Supported by SCTP

The TCP and SCTP protocols are both connection-oriented protocols that have built-in keepalives of the same kind, as described in the previous section.

Keepalives are implemented in SCTP using two special chunk types called HEARTBEAT and HEARTBEAT ACKNOWLEDGMENT, which are described in the following sections.

Heartbeat Implementation: SCTP uses the same keepalive timeout as TCP does, and the same default value of two hours. However, it does not use the same mechanism as TCP does to implement keepalives. Instead, it uses a special chunk called the HEARTBEAT chunk. When a SCTP endpoint sends a HEARTBEAT chunk to its peer, the peer must respond with a HEARTBEAT ACKNOWLEDGMENT chunk. This allows the sender to determine that the connection is still active. The HEARTBEAT chunk can also contain optional information, such as the current time, which can be used by the sender to determine the round-trip time to the peer.

Heartbeat Implementation Guidelines: The SCTP specification provides guidelines for implementation of the HEARTBEAT chunk. These include: only the endpoint that initiated the connection should send HEARTBEAT chunks; the HEARTBEAT chunk should be sent only if no other chunks have been received for a specified period; HEARTBEAT chunks should be sent to each IP address of the peer; and the HEARTBEAT chunk should be sent to a particular peer address only if it is not in the inactive state.

Heartbeat Implementation Example: Consider a SCTP endpoint that initiates a connection with a peer. The endpoint sends a HEARTBEAT chunk to the peer. The peer responds with a HEARTBEAT ACKNOWLEDGMENT chunk. The endpoint then waits for two hours before sending another HEARTBEAT chunk.

Heartbeat Implementation Example Conclusion: The endpoint sends a HEARTBEAT chunk every two hours to the peer to determine if the connection is still active.

Heartbeat Implementation Example Conclusion Continued: The endpoint sends a HEARTBEAT chunk every two hours to the peer to determine if the connection is still active.

Using Keepalives

Keepalives Supported by SCTP

Heartbeat Chunk Type

Heartbeat Chunk Parameters

Heartbeat Acknowledgment Chunk Type

Heartbeat Acknowledgment Chunk Parameters

Heartbeat Acknowledgment Chunk Error Indication

Heartbeat Acknowledgment Chunk Error Indication Parameters

Heartbeat Implementation

Heartbeat Implementation Guidelines

Heartbeat Implementation Example

Heartbeat Implementation Example Conclusion

Heartbeat Implementation Example Conclusion Continued

Heartbeat Chunk Type

The HEARTBEAT chunk is used by a SCTP endpoint to send a HEARTBEAT chunk to its peer. The HEARTBEAT chunk contains a variable-length parameter field, which can be used to store information such as the current time, which can be used by the sender to determine the round-trip time to the peer.

The HEARTBEAT chunk has the following structure:

```
Chunk Type = 4
Chunk Flags = 0x00
Chunk Length = 4 + Length of Heartbeat Info Parameter
Heartbeat Info Parameter
```

Heartbeat Chunk Parameters

A Heartbeat Info parameter can be sent in a HEARTBEAT chunk. The Heartbeat Info parameter is a variable-length parameter that can be used to store information such as the current time, which can be used by the sender to determine the round-trip time to the peer. The Heartbeat Info parameter has the following structure:

```
Parameter Type = 1
Parameter Length = 4 + Length of Heartbeat Info
Heartbeat Info
```

Heartbeat Acknowledgment Chunk Type

The HEARTBEAT ACKNOWLEDGMENT chunk is used by a SCTP endpoint to send a HEARTBEAT ACKNOWLEDGMENT chunk to its peer. The HEARTBEAT ACKNOWLEDGMENT chunk has the following structure:

```
Chunk Type = 5
Chunk Flags = 0x00
Chunk Length = 4 + Length of Heartbeat Info Parameter
Heartbeat Info Parameter
```

Heartbeat Acknowledgment Chunk Parameters

The HEARTBEAT ACKNOWLEDGMENT chunk can also contain optional information, such as the current time, which can be used by the sender to determine the round-trip time to the peer. The HEARTBEAT ACKNOWLEDGMENT chunk has the following structure:

```
Parameter Type = 1
Parameter Length = 4 + Length of Heartbeat Info
Heartbeat Info
```

Heartbeat Acknowledgment Chunk Error Indication

The HEARTBEAT ACKNOWLEDGMENT chunk can also contain optional information, such as the current time, which can be used by the sender to determine the round-trip time to the peer. The HEARTBEAT ACKNOWLEDGMENT chunk has the following structure:

```
Parameter Type = 2
Parameter Length = 4 + Length of Error Indication
Error Indication
```

Heartbeat Acknowledgment Chunk Error Indication Parameters

The HEARTBEAT ACKNOWLEDGMENT chunk can also contain optional information, such as the current time, which can be used by the sender to determine the round-trip time to the peer. The HEARTBEAT ACKNOWLEDGMENT chunk has the following structure:

```
Parameter Type = 3
Parameter Length = 4 + Length of Error Indication
Error Indication
```

Heartbeat Implementation

The HEARTBEAT chunk is used by a SCTP endpoint to send a HEARTBEAT chunk to its peer. The HEARTBEAT chunk contains a variable-length parameter field, which can be used to store information such as the current time, which can be used by the sender to determine the round-trip time to the peer. The HEARTBEAT chunk has the following structure:

```
Chunk Type = 4
Chunk Flags = 0x00
Chunk Length = 4 + Length of Heartbeat Info Parameter
Heartbeat Info Parameter
```

Heartbeat Implementation Guidelines

The HEARTBEAT chunk is used by a SCTP endpoint to send a HEARTBEAT chunk to its peer. The HEARTBEAT chunk contains a variable-length parameter field, which can be used to store information such as the current time, which can be used by the sender to determine the round-trip time to the peer. The HEARTBEAT chunk has the following structure:

```
Chunk Type = 4
Chunk Flags = 0x00
Chunk Length = 4 + Length of Heartbeat Info Parameter
Heartbeat Info Parameter
```

Heartbeat Implementation Example

The HEARTBEAT chunk is used by a SCTP endpoint to send a HEARTBEAT chunk to its peer. The HEARTBEAT chunk contains a variable-length parameter field, which can be used to store information such as the current time, which can be used by the sender to determine the round-trip time to the peer. The HEARTBEAT chunk has the following structure:

```
Chunk Type = 4
Chunk Flags = 0x00
Chunk Length = 4 + Length of Heartbeat Info Parameter
Heartbeat Info Parameter
```

Heartbeat Implementation Example Conclusion

The HEARTBEAT chunk is used by a SCTP endpoint to send a HEARTBEAT chunk to its peer. The HEARTBEAT chunk contains a variable-length parameter field, which can be used to store information such as the current time, which can be used by the sender to determine the round-trip time to the peer. The HEARTBEAT chunk has the following structure:

```
Chunk Type = 4
Chunk Flags = 0x00
Chunk Length = 4 + Length of Heartbeat Info Parameter
Heartbeat Info Parameter
```

Heartbeat Implementation Example Conclusion Continued

The HEARTBEAT chunk is used by a SCTP endpoint to send a HEARTBEAT chunk to its peer. The HEARTBEAT chunk contains a variable-length parameter field, which can be used to store information such as the current time, which can be used by the sender to determine the round-trip time to the peer. The HEARTBEAT chunk has the following structure:

```
Chunk Type = 4
Chunk Flags = 0x00
Chunk Length = 4 + Length of Heartbeat Info Parameter
Heartbeat Info Parameter
```