SoftProbe

User's Guide

Version 2.2

Created by Mehdi Sotoodeh
With thanks to Brian Larsen for documentation.

mehdisotoodeh@gmail.com

INTRODUCTION 5
MINIMUM SYSTEM REQUIREMENTS 5
INSTALLATION 5
LOADING SOFTPROBE 5
ACTIVATING SOFTPROBE 6
SOFTPROBE'S SCREEN 7
THE STACK WINDOW 8
WATCH WINDOW 9
MEMORY WINDOW 10
REGISTER WINDOW 11
THE CODE WINDOW 12
THE COMMAND/STATUS LINE 12
CONFIGURING AND LOADING SOFTPROBE 13
CONFIGURING AND CREATING BOOTABLE MEDIA 16
COMMANDS 17
COMMAND OVERVIEW 17
COMMAND DESCRIPTIONS 21
DATA WINDOW 22
SET DATA WINDOW: D 22
SCROLL DATA WINDOW: UP, DN, ^UP, ^DN, ~UP, ~DN, RT, LT, ~RT, ~LT 22
DISPLAY SPECIAL ADDRESSES IN DATA WINDOW: ~D, ^D, ~^D, @D, @K, ~@K 23
DATA MARKERS: 0..9, ~0..9 24
DATA WINDOW DISPLAY FORMATS: J, @J 25
CODE WINDOW 25
SET CODE WINDOW: U 25
SCROLL CODE WINDOW: PGUP, PGDN, ^PGUP, ^PGDN, @PGUP, @PGDN, ~PGUP, ~PGDN 25
DISPLAY SPECIAL ADDRESSES IN CODE WINDOW: ~U, ^U, ~^U, @U, ^K, ~^K 26
GOTO CURRENT CS:IP: ./END 27
CODE MARKERS: ^0..9, ~^0..9 27
STACK WINDOW 27
STACK OPERATIONS: K, ^K, ~K, @K, ~@K 27
WATCH WINDOW 28
WATCHES: W, ~W, @W 28
WATCH WINDOW DISPLAY FORMATS: ~J, @J 29
MEMORY 30

EDIT: E 30
FILL: ~M 30
COMPARE: ~C 31
COPY: C 32
FIND: ~F 32
FIND NEXT: F 32
REPLACE: ^R 33
WORKING MEMORY READ: Y 33
WORKING MEMORY WRITE: ~Y 33

REGISTERS 34

REGISTER OPERATIONS: R, ~R/F2, @R, ~@R 34

BREAKPOINTS 35

BREAKPOINT OPERATIONS: B/F9, ~B, ^B, ~^B, @B 35
LIST BREAKPOINTS: L 36

HOOKS 37

HOOK OPERATIONS: H, ~H, ^H, ~^H, @H 37
LIST HOOKS: ~L 38
TIMERTICK WATCH: ~F1 39

EXECUTION 40

GO: G/Q/F5 40
GO HERE: ~G/~Q 41
SET CS:IP HERE: ^G 41
GO FROM HERE: ~^G 41
TRACE 1: T/+/F8 41
TRACE N: ~T 41
SET TRACE COUNT N: ~^T 42
TRACE UP: ^T 42
TRACE HERE: @T 42
STEP: S/-/F10 42
ASSEMBLE: A 43
TERMINATE APPLICATION: ^@Q 43

INTERRUPTS 44

LIST INTERRUPTS: N, ~N 44
MODIFY: @N, ~@N 44

I/O 45

INPUT: I, ~I 45
OUTPUT: O, ~O 45

MACROS 45

MACRO OPERATIONS: @0..9, ~@0..9 45

SYSTEM 46

HELP WINDOW: F1/? 46
USER SCREEN: V/F4 46

REFRESH SOFTPROBE WINDOW: F3 46
CALCULATOR: X, ~X 46
PRINT: ~P, ~^P, ~@P 47
CURRENT TASK: ~F7 48
SHOW MEMORY MAP: ~F10 48
LIST DEVICE DRIVERS: ~F9 48
LIST OPEN FILES: ~F8 49

USING CUSTOM HOOKS 49

METHOD 1 50
METHOD 2 51

Introduction

SoftProbe is a system level debugging/snooping tool for MSDOS that seamlessly hides itself in the background until it is invoked. SoftProbe is designed such that it imposes minimum impact on the system resources while tries to be transparent to other applications and even debuggers. It can be running in the background and invoked whenever needed.
Using SoftProbe, you are able to examine and manipulate memory, set breakpoints, trap interrupts, trace code and perform a variety of other operations mostly with a single key stroke. These operations are discussed in the "COMMANDS" sections of this document.
With SoftProbe you can debug applications, device drivers and with BootProbe you can create bootable floppies or CDs that can debug system boot of the operating system.

Minimum System requirements

- Intel 386 and above
- DOS 3.1+ for SOFTPROB.EXE (TSR/DEVICE DRIVER version) and BOOTPROB.EXE (create/configure bootable disks, No O.S. required at boot time).
- Monochrome or Color Display (optional second monitor)
- 32K RAM + the amount you wish to allocate for "working" memory.
- Hard Disk (0.5MB free)

Installation

Please read the README.TXT file before installation.
Extract and copy the files onto a directory of your choice. The following example assumes that drive C: is your hard disk and that drive A: contains the SoftProbe disk:

Loading SoftProbe

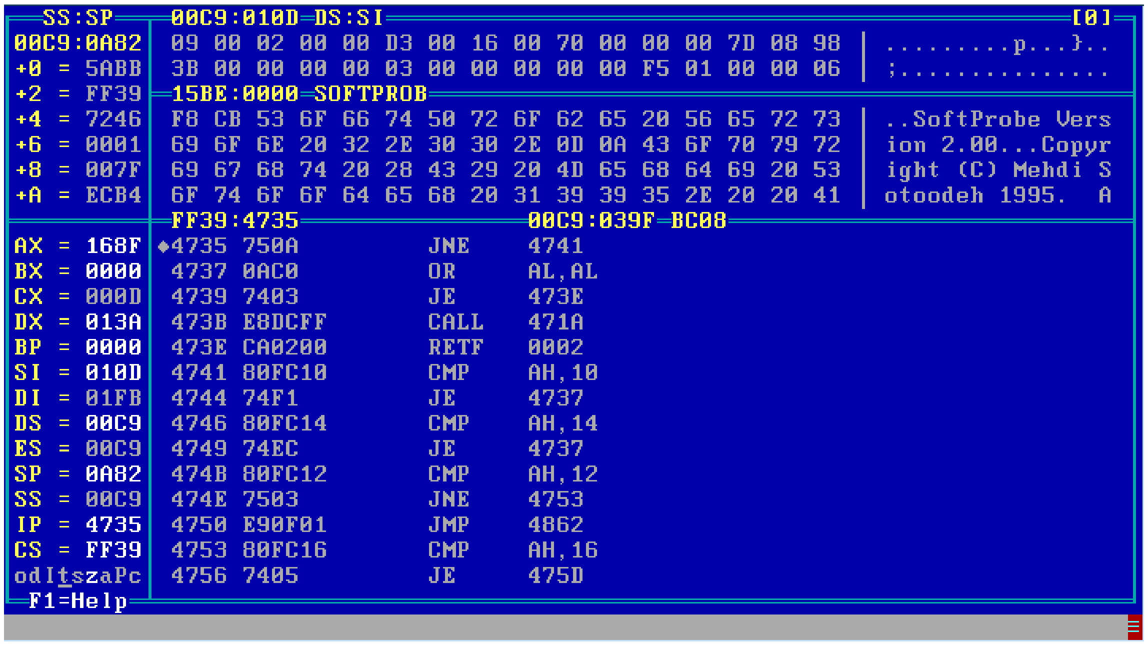
SoftProbe can be loaded in 3 different ways:

1. From MS-DOS prompt, a batch file or AUTOEXEC.BAT.
   C:\SOFTPROB\SOFTPROB.EXE [parameters]
   This will load SoftProbe as a TSR.
2. To load SoftProbe as a device driver, you need to add a line to your CONFIG.SYS, for example:
   DEVICE=C:\SOFTPROB\SOFTPROB.EXE /MEM=100
   Use this method for debugging DOS device drivers.
3. Insert the bootable SoftProbe disk or CD and reset the machine.
   Use this method to take control of the machine before any operating system is loaded.
   You can debug boot managers, check the boot process, diagnose partition sector

problems, or even check for boot sector viruses....
You can create the bootable disk or CD images by using the BOOTPROB.EXE file, which are described in the section "CONFIGURING AND LOADING SOFTPROBE".

Activating SoftProbe

Executing SOFTPROB.EXE loads SoftProbe into memory but doesn't activate it. When activated, you will see following screen:



```
  SS:SP    00C9:010D DS:SI                                              [0]
00C9:0A82  09 00 02 00 00 D3 00 16 00 70 00 00 00 7D 08 98 | .........p...}..
+0 = 5ABB  3B 00 00 00 00 03 00 00 00 00 F5 01 00 00 06    | ;..............
+2 = FF39  15BE:0000 SOFTPROB
+4 = 7246  F8 CB 53 6F 66 74 50 72 6F 62 65 20 56 65 72 73 | ..SoftProbe Vers
+6 = 0001  69 6F 6E 20 32 2E 30 30 2E 0D 0A 43 6F 70 79 72 | ion 2.00...Copyr
+8 = 007F  69 67 68 74 20 28 43 29 20 4D 65 68 64 69 20 53 | ight (C) Mehdi S
+A = ECB4  6F 74 6F 6F 64 65 68 20 31 39 39 35 2E 20 20 41 | otoodeh 1995.  A
           FF39:4735                 00C9:039F BC08
AX = 168F  ◆4735 750A         JNE   4741
BX = 0000   4737 0AC0         OR    AL,AL
CX = 000D   4739 7403         JE    473E
DX = 013A   473B E8DCFF       CALL  471A
BP = 0000   473E CA0200       RETF  0002
SI = 010D   4741 80FC10       CMP   AH,10
DI = 01FB   4744 74F1         JE    4737
DS = 00C9   4746 80FC14       CMP   AH,14
ES = 00C9   4749 74EC         JE    4737
SP = 0A82   474B 80FC12       CMP   AH,12
SS = 00C9   474E 7503         JNE   4753
IP = 4735   4750 E90F01       JMP   4862
CS = FF39   4753 80FC16       CMP   AH,16
odItszaPc   4756 7405         JE    475D
 F1=Help
```

This screen is described in the section "SOFTPROBE'S SCREEN".

SoftProbe is activated in several ways:

- Hold down Left-Shift and then press Right-Shift (if you press Right-Shift first, SoftProbe will NOT be activated!).

- Hold down CTRL and Left-Shift and then press Right-Shift (CTRL and Left-Shift must both be held down before Right-Shift, or SoftProbe will NOT be activated!).
  The only difference between this method and the first method is this method forces the video display into text mode before displaying the SoftProbe screen. The SoftProbe screen will only display properly in text mode. A side effect of this method is that when you return to normal system processing, the user screen remains in text mode. (Use a second monitor for debugging graphic applications).

- The SoftProbe software interrupt occurs while executing a program. This is normally INT 60h, but can be changed with the command line option /INT=<n>. (See the section:

"CONFIGURING AND LOADING SOFTPROBE"). Normally you include this break instruction into your executable.

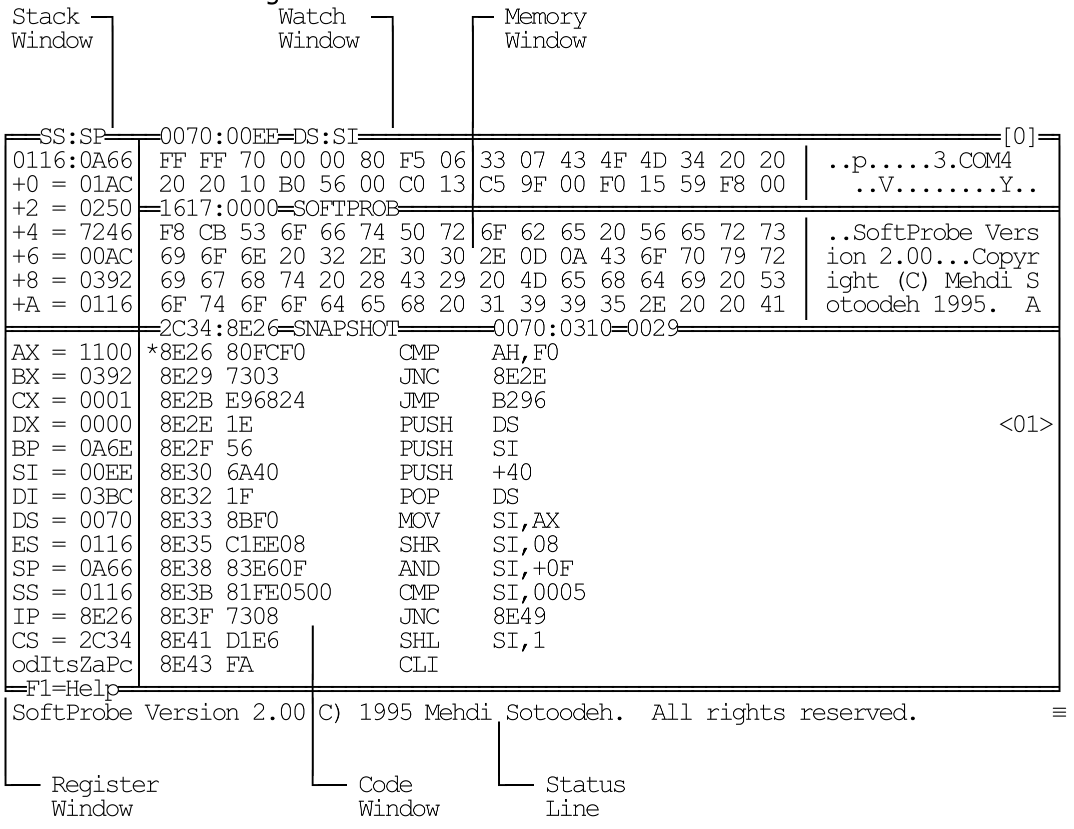
- A breakpoint is encountered during execution. Breakpoints are described in the section "BREAKPOINTS".
- An interrupt "hook" occurs during execution. Hooks are described in the section "HOOKS".

To return to normal system processing press "G" (or "Q" or F5).

SOFTPROBE'S SCREEN

This section describes the screen that is displayed when SoftProbe is activated. Each part of this screen is identified and discussed, familiarity with the SoftProbe screen will be helpful when reading the section "SOFTPROBE COMMANDS".

A sample SoftProbe screen is shown below with each major area of the layout identified. These areas are discussed in greater detail in its related section.



```
╔═SS:SP═══╦0070:00EE═DS:SI═══════════════════════════════════════════════[0]═╗
║0116:0A66║ FF FF 70 00 00 80 F5 06│33 07 43 4F 4D 34 20 20 │ ..p.....3.COM4  ║
║+0 = 01AC║ 20 20 10 B0 56 00 C0 13│C5 9F 00 F0 15 59 F8 00 │   ..V.......Y.. ║
║+2 = 0250╠═1617:0000═SOFTPROB═════════════════════════════════════════════════╣
║+4 = 7246║ F8 CB 53 6F 66 74 50 72│6F 62 65 20 56 65 72 73 │ ..SoftProbe Vers║
║+6 = 00AC║ 69 6F 6E 20 32 2E 30 30│2E 0D 0A 43 6F 70 79 72 │ ion 2.00...Copyr║
║+8 = 0392║ 69 67 68 74 20 28 43 29│20 4D 65 68 64 69 20 53 │ ight (C) Mehdi S║
║+A = 0116║ 6F 74 6F 6F 64 65 68 20│31 39 39 35 2E 20 20 41 │ otoodeh 1995.  A║
╠═════════╬═2C34:8E26═SNAPSHOT═══════0070:0310═0029══════════════════════════╣
║AX = 1100║*8E26 80FCF0       CMP    AH,F0                                     ║
║BX = 0392║ 8E29 7303         JNC    8E2E                                      ║
║CX = 0001║ 8E2B E96824       JMP    B296                                      ║
║DX = 0000║ 8E2E 1E           PUSH   DS                                    <01>║
║BP = 0A6E║ 8E2F 56           PUSH   SI                                        ║
║SI = 00EE║ 8E30 6A40         PUSH   +40                                       ║
║DI = 03BC║ 8E32 1F           POP    DS                                        ║
║DS = 0070║ 8E33 8BF0         MOV    SI,AX                                     ║
║ES = 0116║ 8E35 C1EE08       SHR    SI,08                                     ║
║SP = 0A66║ 8E38 83E60F       AND    SI,+0F                                    ║
║SS = 0116║ 8E3B 81FE0500     CMP    SI,0005                                   ║
║IP = 8E26║ 8E3F 7308         JNC    8E49                                      ║
║CS = 2C34║ 8E41 D1E6         SHL    SI,1                                      ║
║odItsZaPc║ 8E43 FA           CLI                                              ║
╚═F1=Help═╩══════════════════════════════════════════════════════════════════╝
 SoftProbe Version 2.00 C) 1995 Mehdi Sotoodeh.  All rights reserved.         ≡
```

The Stack Window

This window displays a portion of the stack, you are able to control whether the display starts at SS:SP or SS:BP. This is the only control you have over this window, if you need more control, you can copy this window to the Data window with the "~^d" (Shift-CTRL-d) command.

As SP (or BP) changes, the Address and Stack Data are automatically updated.

```
                   Register pair
                   being watched
                   (SS:SP or SS:BP)
                         │
                     ┌───┴───┐
                     │       │

                  ╒══SS:SP═══╕  ┌─ Address (value of SS:SP or SS:BP)
                  │0116:0A66 ├──┘
 Offset from   ┌──│+0 = 01AC ├──┐ «─── Data at top of stack (SS:SP or SS:BP)
 SS:SP or      │  │+2 = 0250 ╞  │
 SS:BP      ───┤  │+4 = 7246 │  ├─ 12 words of Stack data
               │  │+6 = 00AC │  │
               │  │+8 = 0392 │  │
               └──│+A = 0116 ├──┘
                  ╞══════════╡
                  │          │
```

Watch Window

This window displays a portion of memory in several formats, which are controllable by you: BYTES (8 bits), WORDS (16 bits), DWORDS (32 bits) or ASCII (8 bits). You can also specify 8 watches, each of which may be an address, a register pair or a combination of the two. Although SoftProbe remembers all 8 watches, it displays only 1 at a time. You can select the watch to display by using the commands "W" or "~w" (Shift-w).

You can change the watch expression or select which watch to display. You can not directly scroll this window; however, the window is automatically adjusted so that the upper left corner always shows the contents of the current watch address. For example if the watch involves a register, this register may change during execution of a program, which causes the address and data of the watch window to change as well.

At the top of this window you will see the current watch address, the current watch expression and the current watch number. The watch address is the address that the watch expression resolves to. The current watch number shows which of the watches (0..7) is currently displayed. The ASCII representation may be shown as is or with an "ASCII filter", which limits the display to ASCII values 20h..7Eh.

```
                  Watch expression,                                     Current
                  combination of                                        watch
Watch address     registers & offsets                                   number
      │                 │                                                  │
  ┌───┴────┐ ┌──────────┴──────┐                                          ┌┴┐
╒═0070:00EE═DS:SI═════════════════════════════════════════════════════════[0]═╕
│ FF FF 70 00 00 80 F5 06 33 07 43 4F 4D 34 20 20 │ ..p.....3.COM4          │
│ 20 20 10 B0 56 00 C0 13 C5 9F 00 F0 15 59 F8 00 │   ..V........Y..        │
  └───────────────────────┬───────────────────────┘  └─────────┬─────────┘
                          │                                    │
                      hex values                          ASCII values
```

Display examples:

WORD format:

```
╒═36B8:1238═ES:BX+SI+1234═════════════════════════════════════════════════[3]═╕
│ 1238 3231 3433 3635 3837 3039 0000 0000 0000     │ 1234567890......        │
│ 1248 0000 0000 0000 0000 0000 0000 0000 0000     │ ................        │
  └─┬┘
    │
  offset
```

DWORD format:

```
╒═5209:0000═ES:BX═════════════════════════════════════════════════════════[2]═╕
│ 0000 34333231  38373635  00003039  00000000      │ 123456789.......        │
│ 0010 00000000  00000000  00000000  00000000      │ ................        │
  └─┬┘
    │
  offset
```

ASCII format:

```
══5B96:730B═5B96:BX+SI+6723═════════════════════════════════════════════════[7]══╗
│ ABCDEFGHIJKLMNOPABCDEFGHIJKLMNOPABCDEFGHIJKLMNOPABCDEFGHIJKLMNOP               │
│ ABCDEFGHIJKLMNOPABCDEFGHIJKLMNOPABCDEFGHIJKLMNOPABCDEFGHIJKLMNOP               │
```

Memory Window

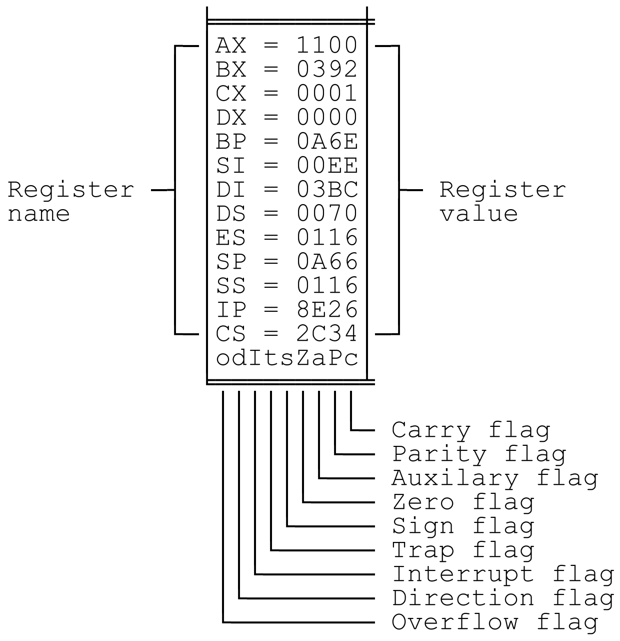
This window is similar to the Watch Window, but you have more control over it. You can set and select 10 markers for this window. You are able to scroll this window with a lot of flexibility. See the section "DATA WINDOW" in the command descriptions.

```
Memory address  Owner of memory
seg:ofs         block, if known

       │          │
   ┌───┴────┐ ┌───┴────┐
╞══1617:0000═SOFTPROB═══════════════════════════════════════════════════════╡
│ F8 CB 53 6F 66 74 50 72 6F 62 65 20 56 65 72 73 │ ..SoftProbe Vers │
│ 69 6F 6E 20 32 2E 30 30 2E 0D 0A 43 6F 70 79 72 │ ion 2.00...Copyr │
│ 69 67 68 74 20 28 43 29 20 4D 65 68 64 69 20 53 │ ight (C) Mehdi S │
│ 6F 74 6F 6F 64 65 68 20 31 39 39 35 2E 20 20 41 │ otoodeh 1995.  A │
```

Register Window

This window shows the registers, their values and the flags. Flags shown in lower case are CLEAR (= 0). Flags shown in upper case are SET (= 1). Flags and registers that are highlighted have changed. Keep in mind that the changes reflect the net result of all instructions executed between the last execution command (step, trace, go, etc.) and the current snapshot.



```
AX = 1100
BX = 0392
CX = 0001
DX = 0000
BP = 0A6E
SI = 00EE
DI = 03BC
DS = 0070
ES = 0116
SP = 0A66
SS = 0116
IP = 8E26
CS = 2C34
odItsZaPc
```

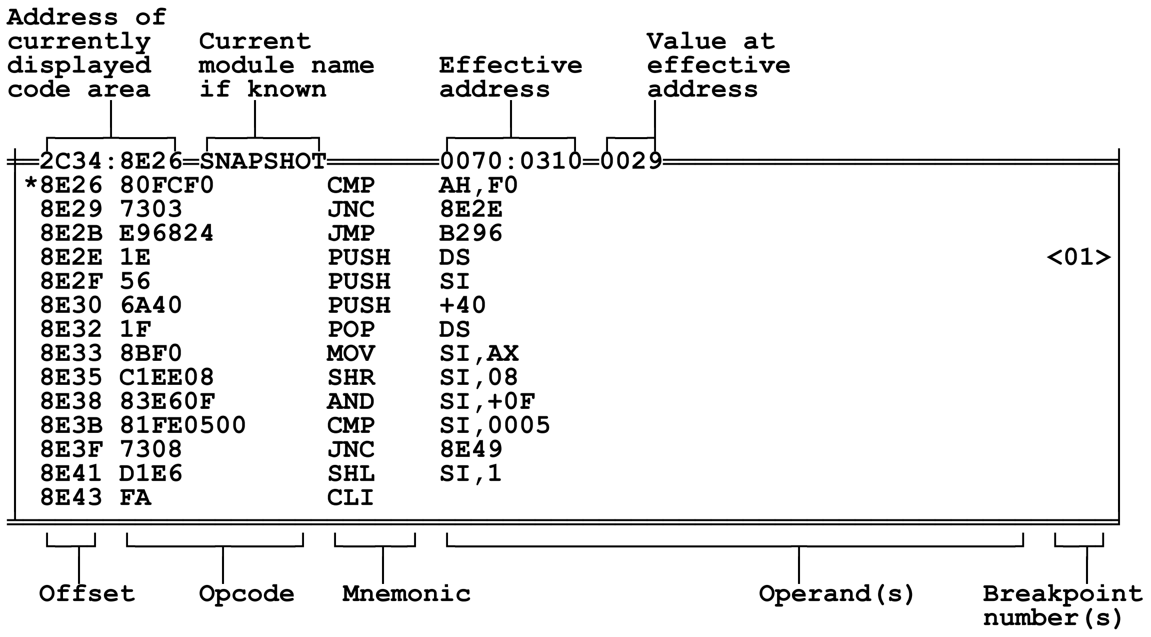
In this example, the Interrupt Flag, the Zero Flag and the Parity Flag are set (value of 1) while the others are clear (value of 0).

The Code Window

The Code window displays a section of code. Normally the code at the current CS:IP is displayed, but you may temporarily view another area by scrolling or by specifying an address. A diamond ("*" is shown here) at the far left of an instruction indicates that instruction is at the current CS:IP.

The opcode column can show a maximum of 6 bytes. It is possible for an instruction to be longer than 6 bytes, in which case these extra bytes are not displayed. This is not normally important, but if needed they can be copied to Data window with the command "~d" (Shift-d).

If any breakpoints are set on instructions displayed in the Code window, their numbers will be displayed at the right edge of the Code window.


Address of currently displayed code area
Current module name if known
Effective address
Value at effective address
Offset
Opcode
Mnemonic
Operand(s)
Breakpoint number(s)

```
══2C34:8E26═SNAPSHOT═══════0070:0310═0029═══════════════════════════════
 *8E26 80FCF0        CMP    AH,F0
  8E29 7303          JNC    8E2E
  8E2B E96824        JMP    B296
  8E2E 1E            PUSH   DS                                     <01>
  8E2F 56            PUSH   SI
  8E30 6A40          PUSH   +40
  8E32 1F            POP    DS
  8E33 8BF0          MOV    SI,AX
  8E35 C1EE08        SHR    SI,08
  8E38 83E60F        AND    SI,+0F
  8E3B 81FE0500      CMP    SI,0005
  8E3F 7308          JNC    8E49
  8E41 D1E6          SHL    SI,1
  8E43 FA            CLI
```

The Command/Status Line

The bottom line of the screen is used to display error messages, status messages or to receive input for commands. The symbol "≡" will be flashing, unless SoftProbe is waiting for additional command input.

Configuring and Loading SoftProbe

Once loaded, SoftProbe remains in memory until the system is rebooted or the DOS session it is running in is terminated. For example if you load SoftProbe in a DOS box under Windows and then close that DOS box, SoftProbe will no longer reside in memory. SoftProbe is comprised of 3 executables:

SOFTPROB.EXE

This is the standard SoftProbe application. It is loaded simply by executing it from the DOS command line. It may also be loaded from AUTOEXEC.BAT or as a device driver in CONFIG.SYS.

SoftProbe can be loaded into high memory.

SOFTPROB.EXE has the following command line options (described later):

```
/MEM=<n>
/INT=<n>
/PRN=<n>
/MLOAD <filename>
/MSAVE <filename>
/LOAD  <filename>
/SAVE  <filename>
/HELP
/?
```

BOOTPROB.EXE

BOOTPROB creates or reconfigures a "bootable" version of SoftProbe images. Created images can be copied to floppy or CD. Booting from this image loads SoftProbe BEFORE the operating system. This lets you debug the operating system as it loads. This could be useful if you want to examine the boot process, diagnosing partition sector, checking for boot viruses or writing your own O.S. or bootable application provided that it uses Real or Virtual 8086 Mode of the CPU.

When you boot your system using BOOTPROB generated image, you will see the SoftProbe copyright and then the following prompt:

```
Loading ....
Boot from ? [A/C]
```

At this point SoftProbe is already loaded. Press "C" to start booting from your hard disk or replace the SoftProbe disk with your bootable disk and press "A" when ready. After this step, SoftProbe will be invoked with a breakpoint at the first instruction of the boot sector. You will now be able to debug the rest of the boot process or let the system continue like it normally would. In some cases when booting from the floppy disk you will be prompted a 2nd time for the drive to boot from. If this occurs, select the same drive again and the boot process should continue normally.

Since this version of SoftProbe is loaded before the operating system, there are some commands which are not available. If you attempt to use one of these commands the following message will appear on the Status line:

Command not supported by BOOTPROB.

BOOTPROB runs on DOS 3.1 or above for the image creation phase.

When BOOTPROB is executed you can use the following command line parameters (explained later):

/MEM=<n>
/INT=<n>
/PRN=<n>
/MLOAD <filename>
/CD
/HELP
/?

SPL.EXE

This is a loader for SoftProbe. If SoftProbe is not loaded, SPL aborts with the error:

SoftProbe is not loaded.

The purpose of SPL is to load an executable program into memory and then invoke SoftProbe so that you can debug the program from the 1st instruction. This is especially useful for applications for which you do not have the source code.
The command line of SPL is as follows:

```
SPL [d:][path\]<progname>[.EXE]  [<parameters>]
    └─────┬─────┘└──────┬───────┘  └─────┬──────┘
          │             │                │
  Optional drive   Program to load   Parameters
  and path of      ".EXE" = default  to pass to
  program to load  extension         progname
```

SPL works with either the standard SOFTPROB.EXE or the version of SoftProbe created by BOOTPROB.EXE.

Below is a description of the command line options for SoftProbe. The short form of each option is shown below the long form:

/MEM=<n>
/M=<n>

Set size of working memory to n paragraphs, the default is 16 paragraphs (256 bytes). This memory is allocated by SoftProbe for your use. You can use it for example to load files, to copy memory, assemble code, etc.
The minimum size is 0, which does not allocate any working memory. If no working memory is allocated, you will not be able to use the Custom feature of the Hook command (H).

	The maximum size depends on the amount of contiguous memory available on your system. Obviously the more memory you allocate for SoftProbe's working memory the less you have for other applications. Normally the default will be sufficient. This option is not allowed if SoftProbe has already been loaded, it may be used only when SoftProbe is first loaded.
/INT=<n> /I=<n>	Use INT n for SoftProbe's break, the default is 60h. This is the interrupt that will invoke SoftProbe, for example if BB is specified, you would use INT 0BBh in an assembly program in order to invoke SoftProbe. This option is not allowed if SoftProbe has already been loaded, it may be used only when SoftProbe is first loaded.
/PRN=<n> /P=<n>	Select LPTn as the printer port, the default is LPT1. Valid values are 1..4. This default is used by the print commands (~p, ~^p, ~@p). This option is not allowed if SoftProbe has already been loaded. It may only be used when SoftProbe is first loaded.
/ MLOAD <filename> / ML <filename>	Redefine all macros (0..9) with the definitions found in the FILENAME. You can use this option to setup your custom macros. This option may be used both when SoftProbe is first loaded and/or after SoftProbe has already been loaded.
/MSAVE <filename> /MS <filename>	Save the macro definitions into the specified file. This file can be reloaded when needed using the /MLOAD option. This option may be used both the when SoftProbe is first loaded and/or after SoftProbe has already been loaded. This option is NOT available when using BOOTPROB.EXE.
/LOAD <filename> /L <filename>	Load the specified file into working memory. This option may be used only AFTER SoftProbe has been loaded. This option is NOT available when using BOOTPROB.EXE.
/SAVE <filename> /S <filename>	Write the contents of the working memory into the specified file. This option may be used only AFTER SoftProbe has been loaded. This option is NOT available when using BOOTPROB.EXE.
/CD	This option is available for BOOTPROB only and when used it creates the image file for bootable-CD’s. If this option is not used, the created image will be the binary representation of a 1.44MB 3.5’ floppy drive.

/HELP, /H, /?	Lists all the options available on SoftProbe's command line.

Configuring and Creating Bootable Media

BOOTPROB can be used to create two different bootable images:

1. Bootable floppy image (FD_BOOT.IMG). This is the default mode and a 1.44MB file is created which is a binary image for a 3.5′ floppy disk. This image can be used for creation of floppies as well as creation of bootable CD's when emulating a floppy drive. This method of CD booting is more common than our second method.
2. Bootable CD image (CD_BOOT.DAT). Generated image

Note that BOOTPROB creates the image file on the current working directory.

You can use third party tools such as RAWRITE2 (see http://en.wikipedia.org/wiki/RaWrite2) to transfer FD_BOOT.IMG images to the physical floppy disks. For CD's you can use pre-mastering tools such as MKISOFS to create bootable ISO images and then use your favorite CD burning tool to create the physical CDs.

Here are sample setting for creation of ISO images for the two modes:

```
mkisofs.exe -b FD_BOOT.IMG -boot-load-seg -o FD_BOOT.ISO .

mkisofs.exe -b CD_BOOT.DAT -no-emul-boot -boot-load-seg 0x7c0 -o CD_BOOT.ISO .
```

COMMANDS

Command Overview

Whenever possible commands are a single keystroke or a keystroke modified by SHIFT, CTRL or ALT. Usually a command that uses information has a simpler keystroke than a command that sets information. The following table shows the available keystrokes. A more detailed description follows the table.

Here are some abbreviations used in the command summary table and the command descriptions:

Up	= The UP arrow key.
Dn	= The DOWN arrow key.
Rt	= The RIGHT arrow key.
Lt	= The LEFT arrow key.
PgUp	= The PgUp key on the keypad.
PgDn	= The PgDn key on the keypad.
HM	= The Home key on the keypad.
EN	= The End key on the keypad.
~	= The Shift key.
^	= The Ctrl key.
@	= The Alt key.

..	= Represents a range, for example:
	0..9 ={0,1,2,3,4,5,6,7,8,9}
	~0..9={~0,~1,~2,~3,~4,~5,~6,~7,~8,~9}

/	= Separates keystrokes having the same function, for example:
F1/?	= "F1" and "?" are identical (both display the Help screen).

Key	No shift	Shift	Ctrl	Ctrl-Shift	Alt	Alt-Shift	Ctrl-Shift
1	Goto data marker 1	Set data marker 1	Goto code marker 1	Set code marker 1	Execute macro 1	Set macro 1	
2	Goto data marker 2	Set data marker 2	Goto code marker 2	Set code marker 2	Execute macro 2	Set macro 2	
3	Goto data marker 3	Set data marker 3	Goto code marker 3	Set code marker 3	Execute macro 3	Set macro 3	
4	Goto data marker 4	Set data marker 4	Goto code marker 4	Set code marker 4	Execute macro 4	Set macro 4	
5	Goto data marker 5	Set data marker 5	Goto code marker 5	Set code marker 5	Execute macro 5	Set macro 5	
6	Goto data marker 6	Set data marker 6	Goto code marker 6	Set code marker 6	Execute macro 6	Set macro 6	
7	Goto data marker 7	Set data marker 7	Goto code marker 7	Set code marker 7	Execute macro 7	Set macro 7	
8	Goto data marker 8	Set data marker 8	Goto code marker 8	Set code marker 8	Execute macro 8	Set macro 8	
9	Goto data marker 9	Set data marker 9	Goto code marker 9	Set code marker 9	Execute macro 9	Set macro 9	
0	Goto data marker 0	Set data marker 0	Goto code marker 0	Set code marker 0	Execute macro 0	Set macro 0	
A	Assemble	Char codes					
B	Toggle Bkpt (B1/B2)	Toggle Bkpt (B2 only)					
C	Copy memory	Memory compare					
D	Set data: seg:offset	Display code as data	Display stack as data	Display watch as data		Display effective address	
E	Edit data memory						
F	Find next text	Define text to find					
G	Go	Go to here	Set CS:IP here	Go from here			
H	Set hook		Enable hook	Disable hook	Remove hook		
I	IN Byte	IN word					
J	Toggle data format	Toggle watch format			Toggle ASCII filter		
K	Toggle stack SS:SP/BP		Display NEAR RET as code	Display FAR RET as code	Display NEAR RET as data	Display FAR RET as data	

Key	No shift	Shift	Ctrl	Ctrl-Shift	Alt	Alt-Shift	Ctrl-Shift
L	List breakpoints	List hooks					
M		Fill memory					
N	List changed interrupts	List all interrupts			Load interrupts	Restore interrupts	
O	OUT byte	OUT word					
P		Print code		Print data bytes		Print Data words	
Q	Go	Go Here					Terminate app
R	Modify registers	List 32-bit registers	Replace text		Compare registers	Snapshot registers	
S	Step						
T	Trace 1	Trace n	Trace up	Set trace n count	Trace here		
U	Display code seg:off	Display data as code	Display watch as code	Display stack as code	Display E.A. as code		
V	Toggle user screen (Bksp/TAB)						
W	Goto next watch	Goto previous watch			Set watch		
X	Hex calculator	Decimal calculator					
Y	Read working memory	Write working memory					
Z							
.	Display code at CS:SP						
?	Help						

Key	No shift	Shift	Ctrl	Ctrl-Shift	Alt	Alt-Shift	Ctrl-Shift
Up	Data up by one line	Data Seg-1	Data up one page				
Down	Data down by one line	Data Seg+1	Data down one page				
Right	Data Off+1	Data normalize					
Left	Data Off-1	Data align paragraph					
Page Up	Code up one line	Code Off-1	Code up one page		Code Seg-1		
Page Down	Code down one line	Code Off+1	Code down one page		Code Seg+1		
Home							
End	Display code at CS:IP						
F1	Help	Timer tick watch					
F2	List 32-bit registers						
F3	Refresh screen						
F4	User screen (Bksp=CLS)						
F5	Go						
F6							
F7		Current task (PSP)					
F8	Trace 1	List open files					
F9	Toggle breakpoint (B1/B2)	List device drivers					
F10	Step	List memory map					
Keypad +	Trace 1						
Keypad -	Step						

Command Descriptions

Some keys have special functions when a command requires additional input. The most common of these keys are shown below; however some commands use them differently. When this occurs it will be documented in the description of that command. If a command description does not mention one or more of these keys it is assumed that these functions are as follows:

ESC	Aborts the command and leaves the original field values intact.
ENTER	Performs the command using whatever values are currently in the input fields.
SPACE	Moves to the next input field. If you are in the last input field the cursor moves to the first field.
TAB	Moves to the next input field. If you are in the last input field the cursor moves to the first field.
~TAB	Moves to the previous input field. Has no effect if you are in the first input field.
+	Increases the value of the current field by 1. If the field contains the maximum value, it wraps to 0.
-	Decreases the value of the current field by 1. If the field contains 0, it wraps to the maximum value.

Scrolling up and down are defined as follows:

UP	The resulting address of the window will have a smaller value. You can think of the window moving UP while the data seen in the window remains stationary or you can think of the window remaining stationary while the data moves DOWN.
DOWN	The resulting address of the window will have a greater value. You can think of the window moving DOWN while the data seen in the window remains stationary or you can think of the window remaining stationary while the data moves UP.

Some commands use a temporary window in order to display information. If this data does not fit on 1 screen, then you can use the following keys to scroll the window:

Up arrow	Scroll window up 1 line.
Down arrow	Scroll window down 1 line.
Page Up	Scroll window up 1 page.
Page Down	Scroll window down 1 page.
Home	Go to top of window.
End	Go to bottom of window.
ESC	Exit window. The window might still be displayed, but it is no longer active (no longer scrollable).

The commands listed below are in the same groupings in which they occur in SoftProbe's help screen. They are:

DATA WINDOW
CODE WINDOW
STACK WINDOW
WATCH WINDOW

REGISTERS
MEMORY
BREAKPOINTS
HOOKS
EXECUTION
INTERRUPTS
I/O
MACROS
SYSTEM

Each command heading identifies the command and list the keystrokes used by that command, for example:

Register Operations: R, ~r/F2, @R, ~@r

After the name of the command (or group of commands) there is a list of keystrokes that invoke the command(s). Keystrokes separated by commas indicate multiple commands. Keystrokes separated by slashes indicate alternative keystrokes for the same command.

In the above example, "Register Operations" is comprised of 4 commands. The first is invoked by "R", the 2nd by either "F2" or "~r", the 3rd by "@R" and the last by "~@r". Remember that "~" represents the Shift key and "@" represents the Alt key.

DATA WINDOW

Set Data window: D

Allows you to change the address of the Data window by changing the values in the segment and offset fields at the upper left of this window.

ENTER, SPACE, UP and DN function identically as follows:
If you are in the segment field the cursor is moved to the offset field.
If you are in the offset field the command is performed.

The TAB and SHIFT-TAB keys are not available in this command.

Scroll Data window: Up, Dn, ^Up, ^Dn, ~Up, ~Dn, Rt, Lt, ~Rt, ~Lt

The Data window may be scrolled in 8 different ways:

Up	Scroll up 1 line of Data. If the data is in BYTE, WORD or DWORD format then 1 line is 16 bytes. If the data is in ASCII format then 1 line is 64 bytes.
Dn	Scroll down 1 line of Data. If the data is in BYTE, WORD or DWORD format then 1 line is 16 bytes. If the data is in ASCII format then 1 line is 64 bytes.

^Up	Scroll up 1 page. This has the same effect as pressing Up 4 times.
^Dn	Scroll down 1 page. This has the same effect as pressing Dn 4 times.
~Up	Scroll up 1 whole segment (segment = segment - 1). The offset remains the same.
~Dn	Scroll down 1 whole segment (segment = segment + 1). The offset remains the same.
Rt	Scroll right 1 byte.
Lt	Scroll left 1 byte.

~Rt Normalize Data address. The new Data address is as follows:

Segment = segment + (offset / 16), the remainder is ignored.
Offset = (offset MOD 16).

Although the address may change it is just another form of the original address. It still references the SAME location in memory!

Here are a few examples, showing the result of using this command:

```
F000:8E26 → F8E2:0006
FFFF:001B → 0000:000B
0000:0100 → 0010:0000
```

~Lt Data Align Paragraph. The new Data address is as follows:

```
Segment = unchanged.
Offset  = offset AND 0FFF0h.
```

The resulting address will be less than or equal to the original address. If the address changes, it will be referencing a different location from the original address (unlike the ~Rt command).

Here are a few examples:

```
F000:8E26 → F000:8E20
FFFF:001B → FFFF:0010
0000:0100 → 0000:0100
```

Display special addresses in Data window: ~d, ^D, ~^d, @D, @K, ~@k

Using these commands you can easily change the address of the Data window to one of several special addresses. The contents of the Data window are changed to reflect the new address.

~d	Copies the Code window to the Data window. Sets the address of the Data window to the address of the Code window.
^D	Copies the Watch window to the Data window. Sets the address of the Data window to the address of the Watch window.
~^d	Copies the Stack window to the Data window. Sets the address of the Data window to the address of the Stack window, which is either SS:SP or SS:BP.
@D	Sets the address of the Data window to the current Effective Address.
@K	Displays code at NEAR return address. Sets the SEGMENT of the Data window to CS and the OFFSET of the Data window to SS:[SP]. The data displayed is the code that would be jumped to if a RET instruction were to be executed at this moment. Note that SS:SP is always used, even if SS:BP is displayed in the Stack window.
~@k	Displays code at FAR return address. Sets the SEGMENT of the Data window to SS:[SP+2] and the OFFSET of the Data window to SS:[SP]. The data displayed is the code that would be jumped to if a RETF instruction were to be executed at this moment. Note that SS:SP is always used, even if SS:BP is displayed in the Stack window.

Data Markers: 0..9, ~0..9

These commands let you save and recall up to 10 Data window addresses. This is convenient if you need to swap the Data window display between several addresses.

~0..9	Sets the specified Data marker to the address currently displayed at the top of the Data window.
0..9	Sets the address of the Data window to that of the specified Data marker.

Setting a marker uses the current address of the Data window therefore you must be viewing the address that you wish to mark. For example, if you want to set a Data Marker to the address currently in DS:SI, then you must first set the address of the Data window to DS:SI and then set the marker.

Once set, a marker remains in effect until you change it or restart SoftProbe (i.e. by terminating a DOS box or rebooting).

The default values of the Data markers are as follows:

1	= 1st byte of working memory.
2	= End of working memory.
3..0	= Address 0000:0000.

Code markers function identically but operate on the Code window. (See "Code Markers" for the keystrokes).

Data Window Display Formats: J, @J

You are able to customize 2 aspects of the Data window display.

J	Each time this command is executed the data format cycles to the next format: Byte → Word → Dword → ASCII → Byte...
@J	Toggles the ASCII filter on or off. This affects the ASCII displays of both the Data and Watch windows. When off, all ASCII characters are displayed unmodified. When on, only the ASCII characters between 20h and 7Eh are displayed normally. Other values (00h..1Fh and 7Fh..FFh) are shown as a "." character.

CODE WINDOW

Set Code window: U

Allows you to change the address of the Code window by changing the values in the segment and offset fields at the upper left of this window.

ENTER, SPACE, UP and DN function identically as follows:
- If you are in the segment field the cursor is moved to the offset field.
- If you are in the offset field the command is performed.

The TAB and SHIFT-TAB keys are not available in this command.

Scroll Code window: PgUp, PgDn, ^PgUp, ^PgDn, @PgUp, @PgDn, ~PgUp, ~PgDn

The Code window may be scrolled in 8 different ways:

PgUp	Scroll up 1 line of code. The start of previous instruction will now be at the top of the Code window. The number of bytes scrolled depends on the number of bytes in the previous instruction.
PgDn	Scroll down 1 line of code. The start of next instruction will now be at the top of the Code window. The number of bytes scrolled depends on the number of bytes in the current instruction.

^PgUp	Scroll up 1 page. This has the same effect as pressing PgUp 14 times.
^PgDn	Scroll down 1 page. This has the same effect as pressing PgDn 14 times.
@PgUp	Scroll up 1 whole segment (segment = segment - 1). The offset remains the same.
@PgDn	Scroll down 1 whole segment (segment = segment + 1). The offset remains the same.
~PgUp	Scroll up 1 byte (not necessarily an entire instruction).
~PgDn	Scroll down 1 byte (not necessarily an entire instruction).

Display special addresses in Code window: ~u, ^U, ~^u, @U, ^K, ~^k

Using these commands you can easily change the address of the Code window to one of several special addresses. The contents of the Code window are changed to reflect the new address.

~u	Copies the Data window to the Code window. Sets the address of the Code window to the address of the Data window.
^U	Copies the Watch window to the Code window. Sets the address of the Code window to the address of the Watch window.
~^u	Copies the Stack window to the Code window. Sets the address of the Code window to the address of the Stack window. This will be either SS:SP or SS:BP.
@U	Sets the address of the Code window to the current Effective Address.
^K	Displays code at NEAR return address. Sets the SEGMENT of the Code window to CS and the OFFSET of the Code window to SS:[SP]. The code displayed is the code that would be jumped to if a RET instruction were to be executed at this moment. Note that SS:SP is always used, even if SS:BP is displayed in the Stack window.
~^k	Displays code at FAR return address. Sets the SEGMENT of the Code window to SS:[SP+2] and the OFFSET of the Code window to SS:[SP]. The code displayed is the code that would be jumped to if a RETF instruction were to be executed at this moment. Note that SS:SP is always used, even if SS:BP is displayed in the Stack window.

Goto current CS:IP: ./END

Either of these keys ("." or END) sets the address of the Code window to CS:IP. This does not change CS:IP, it only changes the Code window display. There are other commands that cause this as a side effect, such as a Trace or Step command.

Code Markers: ^0..9, ~^0..9

These commands let you save and recall up to 10 Code window addresses. This is convenient if you need to swap the Code window display between several addresses.

~^0..9	Sets the specified Code marker to the address currently displayed at the top of the Code window.
^0..9	Sets the address of the Code Window to that of the specified Code marker.

Setting a marker uses the current address of the Code window, so you must be viewing the address that you wish to mark. For example if you want to set a Code Marker to the address in SS:BP, then you must first set the address of the Code window to SS:BP and then set the marker.

Once set, a marker remains in effect until you change it or restart SoftProbe (i.e. by terminating a DOS box or rebooting).

The default values of the Code markers are 0000:0000.

Data markers function identically, but operate on the Data window. See "Data Markers" for the keystrokes.

STACK WINDOW

Stack Operations: K, ^K, ~k, @K, ~@k

K	Toggles the address of the Stack window between SS:SP and SS:BP. The data in the window is updated appropriately.
^K	Displays code at NEAR return address. Sets the SEGMENT of the Code window to CS and the OFFSET of the Code window to SS:[SP]. The code displayed is the code that would be jumped to if a RET instruction were to be executed at this moment.

(Note that SS:SP is always used even if SS:BP is displayed in the Stack window).

~^k	Displays code at FAR return address. Sets the SEGMENT of the Code window to SS:[SP+2] and the OFFSET of the Code window to SS:[SP]. The code displayed

is the code that would be jumped to if a RETF instruction were to be executed at this moment.

(Note that SS:SP is always used even if SS:BP is displayed in the Stack window).

@K	Displays code at NEAR return address. Sets the SEGMENT of the Data window to CS and the OFFSET of the Data window to SS:[SP]. The data displayed is the code that would be jumped to if a RET instruction were to be executed at this moment. (Note that SS:SP is always used even if SS:BP is displayed in the Stack window).
~@k	Displays code at FAR return address. Sets the SEGMENT of the Data window to SS:[SP+2] and the OFFSET of the Data window to SS:[SP]. The data displayed is the code that would be jumped to if a RETF instruction were to be executed at this moment. Note that SS:SP is always used, even if SS:BP is displayed in the Stack window.

WATCH WINDOW

Watches: W, ~w, @W

Watches are similar to Data and Code markers but they are manipulated a little differently. There are 8 Watches, the number of the active (currently displayed) watch appears on the top line of the Watch window. The following commands control the Watches:

W	Displays the next watch. If Watch 7 is active then Watch 0 will be displayed.
~w	Displays the previous watch. If Watch 0 is active then Watch 7 will be displayed.
@W	Allows you to redefine the current Watch. Suppose you change Watch 6, the following display will appear on the status line: Watch #06 [ ] You are able to enter a new expression between the brackets. When you press ENTER, the new expression is accepted only if it is valid. If it is invalid then you must fix it or abort by pressing ESC. Once the expression is valid, the address and expression of the Watch window will be updated. When editing a watch expression, the following keys may be used: RT = Move right 1 character without changing anything. LT = Move left 1 character without changing anything. INS = Toggle between Insert and Overwrite modes. HM = Move to the start of the input.

EN   = Move to the end of the input.
DEL = Delete 1 character.
BKSP = Move left and delete 1 character.

When entering an expression, it must be of the form <SEG>:<OFS> as defined below:

SEG = {addr | reg}

OFS = {addr | reg[+reg][+addr]}

addr = A hex number in the range 0..FFFF.
reg = A register: AX,BX,CX,DX,BP,SI,DI,DS,ES,FS,GS,SP,SS,CS,IP.

The following expressions provide examples of each of the possible forms. The addresses and registers used have no significance:

```
0000:046C          = addr:addr
1FDC:AX            = addr:reg
2EC7:CX+1964       = addr:reg+addr
3213:BP+DI         = addr:reg+reg
8E26:ES+GS-000E    = addr:reg+reg-addr

SS:0408            = reg:addr
IP:BX              = reg:reg
DX:SI+1964         = reg:reg+addr
DS:FS+SP           = reg:reg+reg
CS:AX+CX+000E      = reg:reg+reg+addr
```

When SoftProbe is first loaded, the Watches default to the following:
Watch 0 = DS:SI
Watch 1 = ES:DI
Watch 2 = DS:DX
Watch 3 = ES:BX
Watch 4 = SS:SP
Watch 5 = SS:BP
Watch 6 = CS:IP
Watch 7 = ES:SI

Watch Window Display Formats: ~j, @J

You are able to customize 2 aspects of the Watch window display.

~j   Each time this command is executed the data format cycles to the next format:

Byte → Word → Dword → ASCII → Byte...

@J   Toggles the ASCII filter on or off. This affects the ASCII displays of both the Data and Watch windows. When off, all ASCII characters are displayed unmodified. When on, only the ASCII characters between 20h and 7Eh are displayed normally. Other values (00h..1Fh and 7Fh..FFh) are shown as a "." character.

MEMORY

Edit: E

Allows you to edit (modify) memory. The starting address to edit is the current address of the Data window. You'll want to make sure this window is set to the desired address before invoking the Edit command. The Status line becomes an edit field (shown here on 2 lines for convenience):

```
nnnn:nnnn  hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh hh
└───┬───┘  └──────────────────────┬──────────────────────┘
    │                             │
Address to edit             Hex data values (hex input field)

                          |................|
                           └──────┬───────┘
                                  │
                          ASCII data values (ASCII input field)
```

The cursor can be switched between the Hex input field or the ASCII input field by using the TAB or Shift-TAB keys.

The Left and Right arrows move the cursor to the left or right, and will cause the address to scroll if you try to move past either end of the input field. The Up and Down arrows also cause the address to scroll. When the address scrolls, any data that has been entered is accepted.

In order to edit a byte in the Hex input field, enter a hex value from 00..FF and press SPACE (or Left or Right arrow) to move to the next byte. In the ASCII input field type the desired character. The ASCII area accepts only characters whose ASCII value is from 20..7E, if you wish to enter other values you must do so in the Hex input field.

Fill: ~m

Modifies a block of memory, by filling it with a pattern of 1-16 bytes. You are prompted for 2 groups of information, the first prompt is:

```
Fill from nnnn:nnnn To nnnn
          └───┬───┘    └┬┘
              │         │
          Seg:ofs of   Offset of
          1st byte     last byte
          to fill      to fill
```

This specifies the block of memory to fill. If the offset of the last byte is a lower value than the first byte, the fill command does NOT wrap to the top of the current segment, it continues into the next segment, for example:

Fill from 0000:FFFF to 0002

Fills the following bytes: 0000:FFFF, 0001:0000, 0001:0001, 0001:0002.

After you enter the addresses to fill, you are prompted for the data to use:

```
Fill with  hh hh hh hh  ...  hh hh hh hh | ...............
           └─────────────┬─────────────┘   └──────┬───────┘
                         │                        │
           Hex input field, 16 bytes       ASCII input field
```

You can switch between the Hex and ASCII input fields with the TAB key. The data you enter will be used to fill the memory block you defined above. If the memory block is larger than the number of bytes you enter, these bytes will be repeated in sequence until the block is filled. For example, suppose you wish to fill a block of 13 bytes with "ABCD". The result will be:

ABCDABCDABCDA

If the number of characters you enter is larger than the memory block, you will be returned to 1st prompt and have a chance to redefine the memory block size.

Do not worry that the entire hex or ASCII input areas may be filled with data. When you press the ENTER key only the data from the start of the field up to (but not including) the cursor is used. If you move the cursor back to the 1st byte in the field and press ENTER, it has the same effect as pressing ESC (nothing happens).

Compare: ~c

Compares two memory blocks. You are given the following prompt on the Status line:

```
Compare nnnn:nnnn Len nnnn With nnnn:nnnn
        └───┬───┘     └┬─┘      └───┬───┘
            │          │            │
        address 1   length      address 2
```

If the memory blocks being compared contain any differences the Status line will look similar to the following:

```
E011:0010 42     E011:0020 62 | ESC=Cancel
          ─┬─              ─┬─
           │                │
     Hex value at     Hex value at
     1st address      2nd address
```

Any key except the ESC key will show the next difference if there is one. The ESC key will end the command (even if there are more differences).

After the command is completed you will a message like:

nnnnn differeces found.

The number displayed will be the total number of differences found, in DECIMAL. If the number is 00000, then the 2 blocks of memory are identical. If you press ESC before all the differences have been located, then this value is the number of differences found up to the time you pressed ESC.

Copy: C

Copies a block of memory. The Status line will look like this:

```
Copy from nnnn:nnnn - nnnn  To  nnnn:nnnn
          └───┬───┘   └┬─┘      └───┬───┘
              │        │            │
            Source   Source     Destination
            start    end        start
```

Note: instead of a length, you can specify the starting and ending offsets of the block to copy. The destination address may overlap the source address.

Find: ~f

Allows you to specify a character string from 1..16 bytes to be searched for. You can also specify a replacement string if desired. The first prompt is:

```
Find: hh hh hh hh  ...  hh hh hh hh | ................
      └────────────┬──────────────┘   └──────┬───────┘
                   │                         │
      Hex data to find, 16 bytes      ASCII data to find
```

You can switch between the Hex input field and the ASCII input field with the TAB key. Enter the desired data to find, in either the Hex or ASCII input fields. Press ESC to abort or ENTER to move to the following prompt:

```
Replace: hh hh hh hh  ...  hh hh hh hh | ................
         └────────────┬──────────────┘   └──────┬───────┘
                      │                         │
         Hex data to replace, 16 bytes   ASCII data to replace
```

You may press ESC to abort the entire command or you may press ENTER (no replacement data) or you may type in some data and then press ENTER.

The find command only locates the data, it will not make any replacements (to replace data use the "^R" command). If the data is found, the Data window is set to the address of that data. If the data is not found, the message "Not found!" is displayed on the Status line.

Find Next: F

Searches for the next occurrence of the data entered in the Find (~f) command. If the data is found, the Data window is set to the address of the found data. If the data is not found, the message "Not found!" is displayed on the Status line.

Replace: ^R

This substitutes the "find" string with the "replace" string, both of which are defined by the find (~f) command. The substitution will occur at 1 of 2 places:

1) The current address of the Data window. If the find string is there (i.e. a find), then an (~f) command was performed just prior to the replace (^R) command.
2) At the next location that the find string is located. In other words, the replace (^R) command does a find next (F) command and then a replace.

If the find string is not located, then no replacement is performed.

Working Memory Read: Y

This reads (loads) data into SoftProbe's working memory. This memory is 256 bytes by default, but is configurable with the "/MEM=<n>" option when starting SoftProbe.

You will be prompted as follows:

```
Read from address nnnn:nnnn  Len nnnn paragraphs
```

The length to read is in paragraphs (blocks of 16-bytes).

If you specify a length greater than the size of working memory the following message will appear on the Status line:

```
"Invalid Length."
```

otherwise the specified data is copied to the working memory.

This command will not work if you have set the working memory to 0 bytes by using "/MEM=0" on SoftProbe's command line.

Working Memory Write: ~y

This writes data from SoftProbe's working memory to an address that you specify. Typically you will have already used the "Y" command to initialize the working memory. You will be prompted as follows:

```
Write to address nnnn:nnnn  Len nnnn paragraphs
```

The length to write is in paragraphs (blocks of 16-bytes).

If you specify a length greater than the size of working memory the following message will appear on the Status line:

"Invalid Length."

This command will not work if you have set the working memory to 0 bytes by using "/MEM=0" on SoftProbe's command line.

REGISTERS

Register Operations: R, ~r/F2, @R, ~@r

R Allows you to modify one or more registers and/or flags. The cursor will appear over the value of the AX register. Use the Up and Down arrows to move to the desired register field or the flags field.

To modify a register, type in the new value or use "+" and "-". After changing the value you may end the command with ENTER (or ESC) or you may continue to use the Up and Down arrows to move to another register field (or the flags field).

If the cursor is in the flags field, you may use the Left and Right arrows to select the desired flag. The state of the flag is changed by pressing either "+" or "-" (they are identical). After changing the desired flags you may end the command with ENTER (or ESC) or you may continue to use the Up and Down arrows to move to another register field.

If you press ESC, the command is terminated and the current register field (or flags field) is restored to the value it had upon entering that field. Once you move out of a field with ENTER or Up or Down arrow, the value in that field is accepted as the new value.

The ENTER and SPACE keys function identically in this command. The TAB and Shift-TAB keys are not used in this command.

~r/F2 Displays the extended flags and the following extended registers:

EAX,EBX,ECX,EDX,EBP,ESI,EDI,FS,GS,GDT,IDT,MSW.

Other registers are not displayed for 1 of 2 reasons:

1) Their entire contents are already shown in the normal Register window (i.e. DS, ES, SS, SP).

2) They cannot be accessed in Real mode (i.e. LDT).

If bit 0 of the MSW register (Machine Status Word register) is set, then your system is in protected mode (virtual 86), otherwise it is in real mode:

MSW 0011 → protected mode
MSW 0010 → real mode.

You can not modify the information in this window. In addition this command functions on 386 systems or above only.

~@r SoftProbe saves the current values of the registers and flags so you can view these values later. A temporary window will be displayed to the right of the Register window showing the values just saved (naturally they will be identical).

The values saved can not be altered or loaded back into the registers. The main use of this command is so you do not have to write down the register values in order to reference them later. It is the counterpart to the "@R" command.

These values are saved until you perform another "~@r" command or until SoftProbe is unloaded.

@R Compares the current register and flag values with those previously stored with the "~@r" command. A temporary window showing the previously saved values is displayed to the right of the Register window. Any values that are different between the current and saved registers are highlighted in the temporary window.

When SoftProbe is first loaded, the initial values of the saved registers and flags are all 0.

BREAKPOINTS

Breakpoint Operations: B/F9, ~b, ^B, ~^b, @B

Breakpoints can be Set, Removed, Enabled or Disabled. When a breakpoint is set it is automatically enabled. There are 2 types of breakpoints referred to as B1 and B2. B1 is a 1-byte breakpoint that uses INT 3 and B2 is a 2-byte breakpoint that uses INT 60h (or the interrupt you specified by the /INT=<n> parameter when you started SoftProbe).

You can have up to 15 breakpoints numbered 01..0F. When a breakpoint is set it is assigned the lowest available breakpoint number. When a breakpoint is removed its number becomes available, so there may be gaps in the numbering.

B/F9 Toggles the breakpoint at the top instruction of the Code window. If a breakpoint already exists there, then it is removed, otherwise a new breakpoint is set and enabled.

When setting a breakpoint, SoftProbe automatically uses B1 on 1-byte instructions and B2 in all other cases. When removing a breakpoint the type (B1 or B2) is not important.

~b The same as "B" but forces a B2 breakpoint to be set even on 1-byte instructions. If a breakpoint already exists, it is removed regardless of its type (B1 or B2).

^B Enables an existing breakpoint that was previously disabled. You are prompted for the breakpoint number to enable as shown below:

```
Enable Break #00  [0=All]
```

~^b Disables an existing breakpoint. You are prompted for the breakpoint number to disable as shown below:

```
Disable Break #00  [0=All]
```

@B Removes an existing breakpoint whether it is enabled or disabled. You are prompted for the breakpoint number to remove as shown below:

```
Remove Break #00  [0=All]
```

When prompted for a breakpoint number, type a number between 00 and 0F and press ENTER. If you press ESC instead, the command is aborted.

If you try to enable, disable or remove breakpoints when none have been set, then you will see the message:

```
No breakpoints in use.
```

List Breakpoints: L

Displays a temporary window listing both enabled and disabled breakpoints but not breakpoints that have been removed. The following format is used:

```
Break   #nn   aaaa:aaaa   xx
        ─┬─   └───┬───┘   │└─ 1 = 1-byte breakpoint (INT 3).
   ┌─────┘        │       │   2 = 2-byte breakpoint.
 breakpoint  address of   │
 number      breakpoint   └── E = Breakpoint is enabled.
 (00..0F)                     d = Breakpoint is disabled.
```

The example below shows breakpoints #01,#02 and #04, but not #03. This indicates that at one time breakpoints #01..#04 were set, but that #03 was removed. When you remove a breakpoint, the numbers of the other breakpoints are not affected.

```
Break   #01   E1CE:0110   E1  ─── Enabled, 1-byte breakpoint.
Break   #02   E1CE:02A6   d2  ─── Disabled, 2-byte breakpoint.
Break   #04   E1CE:0343   E2  ─── Enabled, 2-byte breakpoint.
```

HOOKS

Hook Operations: H, ~h, ^H, ~^h, @H

Hooks are similar to breakpoints, but allow SoftProbe to gain control when a specific interrupt occurs. You can have up to 8 Hooks numbered 01..08.

H	Sets a Hook. You have several choices when setting a Hook and you will be prompted similarly to the following example (the prompt is slightly different from the real prompt for illustrative purposes, but the fields are the same):

```
Hook 01: INT 21 If (0000≤ AX ≤FFFF) Wait:Y Print:N Setup:N Custom:N
             ──     ─────     ─────       ─        ─        ─        ─
              1       2         3         4        5        6        7
```

The input fields are shown underlined and numbered for easier identification in this discussion:

1. Interrupt number to hook, enter a number from 00..FF.
2. When the interrupt in field 1 occurs, the Hook occurs only if AX is greater than or equal to the value you enter here.
3. When the interrupt in field 1 occurs, the Hook occurs only if AX is less than or equal to the value you enter here.
4. Determines whether or not SoftProbe will pause the program execution when the Hook occurs. If you select "Y" then when the Hook occurs, SoftProbe will display the registers at the top of the user screen and wait for you to press one of the following keys:

ALT	= Invoke SoftProbe AFTER the interrupt has completed.
CTRL	= IGNORE the Hook and continue program execution.
TAB	= Invoke SoftProbe BEFORE any interrupt code executes.

   CS:IP will be at the first instruction within the interrupt, but it has not been executed yet.

5. The register values are sent to the printer. A printer must be available. The default printer is LPT1, but you may change this when SoftProbe is first loaded by using the /PRN=<n> command line option.
6. The Hook verifies that all of SoftProbe's interrupt vectors still point to SoftProbe's routines. If not, they are rewritten to do so. This is useful when running an application that takes control of interrupts that SoftProbe uses, but it must be setup before running such an application. There is a similar option when setting a Timer Tick Watch.

7. The custom option allows you to execute your own code when a Hook occurs, but BEFORE any interrupt code is executed. This is an advanced feature and is documented in the appendix "Using Custom Hooks."

^H Enables an existing Hook that was previously disabled. You are prompted for the Hook number to enable as shown below:

Enable Hook #00 [0=All]

~^h Disables an existing Hook. You are prompted for the Hook number to disable as shown below:

Disable Hook #00 [0=All]

@H Removes an existing Hook whether it is enabled or disabled. You are prompted for the Hook number as shown below:

Remove Hook #00 [0=All]

If SoftProbe is unable to remove the Hook, then the Hook will be disabled instead of removed.

When prompted for a Hook number, type a number between 00 and 0F and press ENTER. If you press ESC instead, the command is aborted.

List Hooks: ~l

Displays a temporary window listing both enabled and disabled hooks, but not hooks that have been removed. The following format is used:

```
Hook  #nn  Int nn  If (yyyy ≤ AX ≤ zzzz)  aaaa:aaaa  xWPSC
      ─┬─  ──┬──     └──────┬──────┘      └───┬───┘  │││││
   ┌───┘     │              │                 │      │││││
   │      Interrupt         │                 │      │││││
 Hook     number     Trap only when       Address    │││││
 number   to hook    AX = yyyy..zzzz      of Int nn  │││││
 (00..08)                                            │││││
                        E = Hook is enabled ─────────┘││││
                        d = Hook is disabled          ││││
                                                      ││││
                Enabled if shown ┌── Wait ────────────┘│││
                Disabled if not──┤   Print ────────────┘││
                shown            │   Setup ─────────────┘│
                                 └── Custom ─────────────┘
```

Here is an example using 2 simple Hooks:

```
Hook  #01  Int 21  If (3C00 ≤ AX ≤ 40FF)  0805:0178  EW
Hook  #02  Int 10  If (0E61 ≤ AX ≤ 0E7A)  0805:0154  d P C
```

Hook #01: Here we are trapping some Int 21h file operations involving file handles: create, open, close, read and write. Their function codes are passed in the AH register: 3Ch, 3Dh, 3Eh, 3Fh and 40h respectively. Since the AL register does not contain input for these operations, it can be any value (00..FF) when the interrupt occurs. This is why the range was set from 3C00h..40FFh, instead of 3C00h..4000h.

This Hook is enabled and the Wait option is being used, we know this from the "EW" at the end of the line.

Hook #02: Here we are trapping the function of the video interrupt that writes a single character to the screen. In this case AL contains the character to display therefore the range we have chosen traps only attempts to write lower case letters (61h..7Ah).

This Hook is disabled and the Print and Custom options have been selected.

TimerTick Watch: ~F1

Allows you to watch memory or ports while your application is running. The watch data is displayed on the top line of the user screen and is updated every timer tick (INT 8). When this command is invoked, the following prompt is displayed on the SoftProbe Status line:

```
F1=Memory  F2=I/O Port  F3-F6=COM1-COM4  F7=Setup  Other keys=Idle
```

Keys other than F1..F7 will cancel the timer tick watch if one is effect.
The keys F1..F7 work as follows:

F1 Sets the watch to the current address of the Data window. The 16 bytes starting from this address are watched. The following data is displayed at the top of the user screen:

```
nnnn:nnnn <16 bytes of data in HEX> <16 bytes of data in ASCII>

 SEG:OFS of data being watched
```

This line will look very similar to the top line of the Data window when it is in byte format. Once the watch is set the address is fixed even if the address of Data window changes.

F2 You are prompted for the port address to watch, enter any value from 0000..FFFF. This port and the next 7 (total of 8) are watched. The following example shows the data displayed at the top of the user screen for port 40h:

```
PORT  0040:B3  0041:13  0042:FF       ....      0047:FF

    1st Port  1st port's data           Last port      Last
    address                             address   port's data
```

F3..F6 Sets the watch to one of the COM ports. The following example shows the information that will be displayed on the top line of the user screen (shown on 2 lines here for convenience):

```
COM1:03F8      BAUD=002400  N 8 1────────────────────┐
   │ └─┬┘      └────┬────┘  │ └─────────┐            │
   │   │            │       │           │            │
 Port  Port     Baud rate   N=none  Data bits    Stop bits
 1..4  address              O=odd   5..8         1, 1.5, 2
                            E=even

DTR=0      RTS=0      DSR=0      CTS=0      DCD=0      RI=0
└─┬─┘      └─┬─┘      └─┬─┘      └─┬─┘      └─┬─┘       └┬┘
  │          │          │          │          │          │
Data       Request    Data Set   Clear        Data      Ring
Terminal   To Send    Ready      To Send   Carrier Indicator
Ready                                       Detect
```

F7 When this option is selected, SoftProbe will verify on every timer tick that all of SoftProbe's interrupt vectors still point to SoftProbe's routines. If not, they are rewritten to do so. This is useful when running an application that takes control of interrupts that SoftProbe uses however, it must be setup before running such an application. There is a similar option when creating a Hook.

EXECUTION

The commands in this section allow you to execute code or to perform related activities. When the SoftProbe screen is redisplayed after doing a Go, Go Here, Trace or Step command, any registers or flags whose values have been modified will be highlighted.

** WARNING ** There are several commands that cause code to execute in trace mode. Normally this is not a problem for normal applications however, since SoftProbe is running in real or V86 mode, you can have unexpected results in some situations (for example code after CLI if it assumes no interrupt or no stack change) or under protected mode when tracing some instructions. The following commands operate in trace mode:

Trace 1, Trace n, Trace Up and Trace Here

Go: G/Q/F5

Executes the current application until one of the following conditions is met:
1) A breakpoint is encountered.
2) A Hook occurs.
3) SoftProbe is invoked by the user (Left-Shift then Right-Shift).

You can think of this as a "quit" command, since until 1 of the above conditions occurs SoftProbe hides itself and the normal user screen is displayed.

Go Here: ~g/~q

Similar to the Go command except a temporary breakpoint is set at the instruction currently at the top of the Code window then the current application is executed. SoftProbe takes control again when one of the following occurs:

1) The instruction at the top of the Code window is encountered.
2) A Hook occurs.
3) SoftProbe is invoked by the user (Left-Shift then Right-Shift).

When ANY of these conditions occur the temporary breakpoint is removed.
If you think that conditions 2 or 3 might occur before reaching the desired instruction, then it's better to set a breakpoint at the desired instruction and do a Go (G) command.

Note that any other breakpoints that you have set will be temporarily ignored until one of the above conditions occurs.

Set CS:IP Here: ^G

Changes the values of CS:IP to point to the instruction currently at the top of the Code window. This is same as doing a Modify Register (R) command and changing CS and IP however it is quicker. This command does not execute any code.

Go From Here: ~^g

This is a combination of the "^G" and "G" commands. First CS:IP is set to the top instruction of the Code window. Then the code is executed starting from that instruction until one of the following occurs:

1) A breakpoint is encountered.
2) A Hook occurs.
3) SoftProbe is invoked by the user (Left-Shift then Right-Shift).

Trace 1: T/+/F8

Executes 1 instruction. Trace will go INTO calls and interrupts.

Trace n: ~t

Same as Trace 1, except n instructions are executed.

Set Trace Count n: ~^t

Defines the value n to be used by the "Trace n" command. You are prompted for the value which must be in 0000..FFFF. Note that 0000 is treated as 65536, the same as when you do a LOOP instruction in Assembly code and CX has an initial value of 0.

Trace Up: ^T

Executes in trace mode until control moves up a level. When this command is issued, the current SS:SP is used as a reference point and code is executed until SS changes or SP takes on a value greater than the reference value. Generally when execution is halted, one of the following will have occurred:

1) POP
2) RET
3) IRET
4) SP was modified directly (for example by an ADD instruction)
5) SS was modified directly (for example by a MOV instruction)

Trace Here: @T

Executes in trace mode until the current instruction is reached. This is mainly useful when you are executing Code and you get the message:

```
Cannot put break here.
```

When this happens you can scroll to the next instruction using PgDn and then use the Trace Here command.

However, you may encounter other situations where this is command is useful.

Step: S/-/F10

Executes 1 "line" of code by placing a temporary breakpoint after the current instruction. All other breakpoints are temporarily disabled until the Step command completes. Step is similar to Trace except for the following instructions:

1) CALL
2) INT
3) LOOP, LOOPE, LOOPZ, LOOPNE, LOOPNZ
4) REP, REPE, REPZ, REPNE, REPNZ

In the following code fragment, assume that the current instruction (CS:IP) is the "CALL" instruction:

```
CALL    2E10:166C    ;Assume the CS:IP is here and you issue a Step
                     ;command.
CMP     AX,1         ;The step command will stop here.
JNE     0923
```

Issuing the Step command will execute all the code inside the CALL and stop at the temporary breakpoint at the CMP instruction. This is true EVEN if you have put a breakpoint inside the code that gets called (for example at address 2E10:166C).

If for some reason the CMP instruction is never executed, the Step command acts like a Go command (G). This can happen if the code called (at 2E10:166C) modifies its return address on the stack.

Assemble: A

Allows you to modify code window using Assembly mnemonics. A working area is displayed on SoftProbe's status line. The address to start assembling at is the current address of the Code window, which may or may not be the same as CS:IP. The status line will look similar to the following:

```
0000:0000 [                                            ] | ESC=Cancel.
```

The "0000:0000" will vary, and is the address to start assembling at.
These keys have the following effect:

ESC	Abort the assemble.
ENTER	Accepts either the instruction just entered or if you typed nothing, accepts the instruction already at the current address. The address following this instruction becomes the next address to assemble at.

If you type an invalid instruction and then press ENTER, the text "ESC=Cancel." will be replaced by "Error...". The address will remain the same and you can edit the instruction. When you press ENTER after typing a valid instruction, this new instruction will be displayed in the code window, which will scroll so that the address of the instruction about to be entered will always be in the window.

When assembling, you may use 32-bit registers (16-bit address), protected mode instructions and floating point instructions.

The working memory is a safe place to assemble some code, since it is set aside for your use. Data marker 1 points to the start of working memory (unless you changed this marker). To assemble at the start of the working memory you could type "1" (display marker 1 in the Data window) then "~u" (copy Data window to Code window) and finally "A" (assemble).

Terminate Application: ^@Q

Attempts to terminate the current application. If successful you will be returned to the parent application (for example DOS). If you are in the middle of a DOS interrupt, the following message will be displayed on the Status line:

```
  Already in DOS. Finish current DOS call and try again.
```

In this case you can set a Hook on Int 21 then issue the Go command ("G") and wait until the Hook occurs. Press ALT to invoke SoftProbe, remove the Hook and then try again to terminate the application.

This command is useful for killing applications that hang as long as SoftProbe is still able to be invoked by pressing Left-Shift then Right-Shift. If the keyboard is disabled, then you are out of luck.

WARNING: It's possible that your system may not be stable after terminating an application. Continuing to operate your system without rebooting may be risky.

INTERRUPTS

List Interrupts: N, ~n

N This window lists the interrupt vectors that have been modified since SoftProbe was started or since @N or ~@n was executed.

~n This window lists all interrupt vectors (00..FF).

Both of these windows have the same format:

```
Int          Was          Now
 nn       aaaa:aaaa    aaaa:aaaa    <name>
───┬───   └────┬────┘  └────┬────┘  └─────  ...  ──┘
   │           │            │                 │
 Interrupt  Old vector   New vector    Name of program that
 number     address of   address of    takes control when the
            Int nn       Int nn        int occurs, if known
```

Modify: @N, ~@n

@N Loads interrupt vectors into SoftProbe's memory. If you now perform the command "N" (list changed interrupts), the list will be empty.

~@n Restores interrupt vectors from SoftProbe's memory. The values restored are either those at the time SoftProbe was loaded or those loaded by the last @N command, which ever occurred last. If you now perform the command "N" (list changed interrupts), the list will be empty. The main use of this command is to disable hooks that are set by TSR programs.

I/O

Input: I, ~i

Inputs (reads) a byte (I) or word (~i) from a port. The prompt is as follows:

```
Input from port 03BC
```

The port address displayed is the last value you entered or 03BC the first time. Enter any port address from 0000..FFFF.

I	Input 1 byte.
~i	Input 1 word (2 bytes).

The data read will be displayed after the prompt. Below is an example for both the "I" and "~i" commands:

```
Input from port 03BC = AB
Input from port 03BC = 7FAB
```

Output: O, ~o

O	Output 1 byte.
~o	Output 1 word (2 bytes).

Outputs (writes) a byte or word to a port. The prompts are as follows:

```
Output AB to port 03BC
Output ABCD to port 03BC
```

The default data is either 00 (0000) or the value you used last time. The default port address is either 03BC or the value you used last time.

Be careful when experimenting, it is possible to hang the system by sending the certain values to certain ports!

MACROS

Macro operations: @0..9, ~@0..9

Macros allow you to record and playback a series of keystrokes.

@0..9	Playback a the specified macro.
~@0..9	Record the specified macro. When macro recording is turned on you will see a flashing "R" in the bottom right corner of SoftProbe's frame, just above the Status line. Macro recording is ended by pressing Shift-ESC.

The total number of character that may be used for all 10 macros combined is 200.

It is valid for a macro to invoke another macro, but macro recording will end as soon as you do so. In other words a macro can invoke another macro (including itself for looping) but only as its LAST operation.

You can SAVE and LOAD macros that you have defined by using two of SoftProbe's command line options:

```
/MLOAD <filename>
/MSAVE <filename>
```

These options are described in the section INTRODUCTION / INSTALLATION.

SYSTEM

Help Window: F1/?

Displays a quick reference window listing the SoftProbe commands and their associated keystrokes. Command descriptions and details are left to the document you are currently reading.

User Screen: V/F4

This command temporarily hides the SoftProbe screen and displays the user screen. The SoftProbe screen is restored by pressing a key.

If the BACKSPACE key is pressed when the user screen is displayed, then the user screen will be reset before restoring the SoftProbe screen.
If the TAB key is pressed when the user screen is displayed, then SoftProbe switches to the secondary display (or back to the primary screen).

Refresh SoftProbe Window: F3

Redraws the entire SoftProbe screen.

Calculator: X, ~x

Invokes a simple calculator on the Status line.

X Accepts input in hex and displays the following prompt:

```
Hex: 0000,0000
```

Valid values for both operands are 0000..FFFF.

~x Accepts input in decimal and displays the following prompt:

```
Dec: 00000,00000
```

Valid values for both operands are 00000..65535. If you try to enter a value greater than 66535 a MOD 65536 is performed on the input.

The only difference between these 2 commands is the type of input they accept. The result of the calculations are the same. these 2 examples produce the same output (the hex and dec input values are equivalent):

```
Hex: ABCD,1234
Dec: 43981,04660
```

The resulting output is:

```
ABCD,1234 +BE01 -9999 *0C374FA4 /0009 %07F9 &0204 |BBFD ^B9F9 [43981,04660]
└───┬───┘                                                       └─────┬─────┘
    │                                                                 │
Input values, always in HEX                            Input values, always in DEC
```

The middle fields are the result of performing various operations using the 2 operands. The above example is used to explain each field:

```
+BE01     : ABCD + 1234 (addition, carry is lost)
-9999     : ABCD - 1234 (subtraction, borrow is lost)
*0C374FA4 : ABCD * 1234 (multiplication, no data is lost)
/0009     : ABCD / 1234 (division, quotient only)
%07F9     : ABCD % 1234 (modulus, remainder only)
&0204     : ABCD & 1234 (bitwise AND).
|BBFD     : ABCD | 1234 (bitwise OR).
^B9F9     : ABCD ^ 1234 (bitwise XOR, Exclusive OR).
```

These calculations do not affect the registers or flags.

Print: ~p, ~^p, ~@p

Sends the specified data to a printer.

~p Print code, using the same format that the Code window uses.

~^p Print data in byte format.

~@p Print data in word format.

These commands have the following format for their prompts:

```
Print 0000:0000 - 0000
      └───┬───┘   └─┬┘
          │         │
     Seg:ofs of  Offset of
     1st byte    last byte
     to print    to print
```

When printing BYTES or WORDS, a multiple of 16 bytes is printed. The number of bytes printed will be greater or equal to the number you specified. For example assume you answered the prompt with the following data:

```
6123:0100 - 0113
```

The last byte printed is the one at offset 011F. This way an entire line is always printed. The next time you print, the prompt is automatically set up to start printing where you left off last time (naturally you can edit these values).

SoftProbe calls the routine that was pointed to by the INT 17 vector when SoftProbe was loaded. Even if the vector itself is changed later, SoftProbe uses the original routine. If you want the printing to go to a network printer, SoftProbe must be loaded after the network is loaded.

Current Task: ~F7

Displays on the Status line, the current PSP and Owner of the code.

Show Memory Map: ~F10

This window lists the applications currently in memory:

```
START      LEN      ENV           PSP       OWNER
 nnnn      nnnn     nnnn          nnnn      <name>
 └─┬┘      └─┬┘     └─┬┘          └─┬┘      └───── . . . ──┘
   │         │        │             │                │
 Starting  Length   Segment       Segment   Path and name of
 Segment   of this  of this       of this   this memory's
 of this   memory   memory's      memory's  owner, if known
 memory    block    environment   PSP
```

The value in the START column locates the start of a particular memory block. For example, if the value 1234 is in the START column the memory block starts at address 1234:0000.

List Device Drivers: ~F9

This window lists the device drivers that are currently loaded:

```
 Header      Stra       Intr       Attr        Device
nnnn:nnnn    nnnn       nnnn       nnnn        xxxxxxxx
└────┬───┘   └─┬┘       └─┬┘       └─┬┘        └───┬───┘
     │         │          │          │             │
Address of   Ofs of     Ofs of     Attributes  Device driver name
the device   strategy   interrupt  of device
header.      routine    routine    driver
```

Device drivers are of 2 main types; Character and Block. Character device headers are an 18 byte structure as follows:

```
LINK    DD 0FFFFFFFFh       ;LINK TO NEXT DEVICE, -1=LAST DEVICE.
ATTR    DW 0000             ;ATTRIBUTES OF DEVICE.
STRA    DW OFFSET STRATEGY  ;OFFSET OF STRATEGY ROUTINE.
INTR    DW OFFSET INTERRUPT ;OFFSET OF INTERRUPT ROUTINE.
DEVICE  DB '        '       ;DEVICE NAME.
```

Block use the same structure except for the DEVICE field. The 1st byte of this field is the number of Block devices associated with this device driver.

A detailed discussion of device drivers is not within the scope of this document. There are numerous reference books available that will provide you with more details.

List Open Files: ~F8

This window lists files that are currently open:

```
FILENAME.EXT  AC      #REF     POSITION   OWNER
xxxxxxxx.xxx  nn      nnnn     nnnnnnnn   nnnn
└──┬───┘ └┬┘  │       └┬┘      └───┬──┘   └┬┘
   │      │   │        │           │       │
  Name and    access  # of opens  File     PSP of the last task
  extension   mode    without a   pointer  that opened the file
  of file             close
```

AC is the access mode that file has been opened with.

Using Custom Hooks

This section documents the "Custom" option used by the Hook command (H). If the custom option is enabled (by entering "Y" in the CUSTOM field), you can specify code to be executed when a Hook occurs. This code will be executed BEFORE any of the code for the hooked interrupt occurs.

The code to be executed must start at the 1st byte of working memory. You can either assemble your code from within SoftProbe or load it using the command:

```
SOFTPROB /Load <filename>
```

Where FILENAME is the name of the file containing the code to load.

When your code is called the following structure will be set up by SoftProbe and is pointed to by SS:BP.

```
IntHookRegStru STRUC
  MI_MON_FLAGS DW ? ; SS:BP    Saved flag values.
  MI_AX        DW ? ; SS:BP+02 ─┐
  MI_BX        DW ? ; SS:BP+04  │
  MI_CX        DW ? ; SS:BP+06  │   Because these registers are already
  MI_DX        DW ? ; SS:BP+08  │   saved before your code is called, your
  MI_BP        DW ? ; SS:BP+0A  ├─ code may safely modify any register
  MI_SI        DW ? ; SS:BP+0C  │   except SS and SP without saving it.
  MI_DI        DW ? ; SS:BP+0E  │
  MI_DS        DW ? ; SS:BP+10  │
  MI_ES        DW ? ; SS:BP+12 ─┘
  MI_RESERVED  DW ? ; SS:BP+14 Used by SoftProbe.
  MI_IP        DW ? ; SS:BP+16 Return address to code that
  MI_CS        DW ? ; SS:BP+18 invoked the INT that was hooked.
  MI_FLAGS     DW ? ; SS:BP+1A Flags of the program that invoked the INT.
IntHookRefStru ENDS
```

In addition the BX register contains the number of the INT that was generated. For example, if BX = 21h, then INT 21h was invoked and hooked. This is useful if you have several hooks and need to know which INT caused the hook.

The layout of "IntHookRefStru" is provided for your information. You should modify these values only if you truly understand what the effect will be.

This feature offers a lot of flexibility but it is processed in 2 main ways:

Method 1

In this method, your code is executed and then the normal code for the hooked interrupt is executed. The algorithm is as follows:

1) The INT instruction occurs for the hooked interrupt and SoftProbe takes control BEFORE any of the code for that interrupt is executed.

2) SoftProbe sets CS:IP to the start of Working Memory (assumed to be your code) and begins executing it. If you wish to debug your code you need to set a breakpoint within your code before the Hook occurs.

Advanced users may wish to modify the values of "IntHookRegStru" at this point in order to change inputs to the hooked interrupt.

3) When your code is finished, you must set CF (carry flag) properly (see step 4). Your code must be terminated by a RETF (far return) instruction in order to give control back to SoftProbe.

4a) Control is returned to SoftProbe. If CF=1 (STC) and the Wait option of the Hook command = "Y" then SoftProbe will now display the registers at the top of the user screen and then wait for your keystroke. (This is what normally happens when a Hook occurs). There are 3 keystrokes available to you:

ALT = Invoke SoftProbe AFTER the interrupt has completed.
CTRL = IGNORE the Hook and continue program execution.
TAB = Invoke SoftProbe BEFORE any interrupt code executes.
CS:IP will be at the first instruction within the
interrupt, but it has not been executed yet.

4b) If CF=0 (CLC) or the Wait option of the Hook command = "N", SoftProbe will not display the registers and will not pause. The normal interrupt code will be executed and the user application will continue to execute until invoked again.

Method 2

In this situation the purpose of your code is to prevent the hooked interrupt from executing. This means you must clean up the stack and return via an IRET instead of a RETF instruction.

1) The INT instruction occurs for the hooked interrupt and SoftProbe takes control BEFORE any of the code for that interrupt is executed.

2) SoftProbe sets CS:IP to the start of Working Memory (assumed to be your code) and begins executing it. If you wish to debug your code you need to set a breakpoint within your code before the Hook occurs.

3) At this point you must clean up the stack, and return to the program that issued the INT that was hooked. A possible way of doing this is shown below:

```
MOV     SP,BP  ;Set SP => top of system stack.
               ;Assumes BP is the same as when your code was called!
POP     AX     ;Dummy POP, removes MI_MON_FLAGS from stack.
               ;You could do an "ADD SP,2" instruction instead.
POP     AX     ;─┐
POP     BX     ; │
POP     CX     ; │
POP     DX     ; │
POP     BP     ; ├─ Restore registers to the values expected by the
POP     SI     ; │  program that invoked the INT that was hooked.
POP     DI     ; │
POP     DS     ; │
POP     ES     ;─┘
ADD     SP,2   ;Throw away MI_RESERVED, since we are returning to
               ;the application instead of to SoftProbe.
IRET           ;Return to program that invoked the hooked INT.
```