SoftProbe/W User's Guide
Version 1.00

Created by Mehdi Sotoodeh
Documentation by Brian Larsen

Mehdi Sotoodeh
msotoodeh@hotmail.com

Introduction
SoftProbe/W (SPW) is a system level debugging/monitoring tool that seamlessly hides itself in the background until it is invoked. SPW started with a small set of functions as a monitoring tool and features were added as it became more mature. Minimum impact on the system (i.e. RAM usage and transparency to other applications and even debuggers) was one of the main goals during the development of SPW. While using SPW, you are able to examine and manipulate memory and registers, set breakpoints, trap interrupts, trace code and perform a variety of other operations. These operations are fully discussed in the *SPW Commands* sections of this document.

System requirements
Intel 80386 CPU or above (SPW runs in protected mode, ring 0).
Windows 3.x or Windows95.
Mono or VGA monitor. Optionally a second monitor on the same or remote system.
?? K RAM above the minimum required to operate Win3.x or Win95.
Hard Disk, 0.5MB of free space.

Installation
Simply copy the files from the SPW disk onto your hard disk. The following example assumes that drive C: is your hard disk and that drive A: contains the SPW disk:

C:	(Select the hard disk)
CD \	(Change to the root directory)
MD SPW	(Make a new directory on your hard disk called "SPW")
CD SPW	(Select the new directory)
COPY A:*.* C:*.*	(Copy all files from the SPW disk to your hard disk)

To install the files onto a different drive or directory just make the appropriate substitutions to the above example.

Loading SPW

SPW is loaded by executing **spw31.exe** for Windows 3.x or **spw95.exe** for Windows 95. The term SPWxx will be used generically to refer to these two programs. SPWxx must be executed from **pure** DOS, which will load SPW and then start Windows. You can not start SPWxx from a DOS Box.

SPW uses the file **spwxx.ini**, which is discussed in more detail in *Configuring SPW*. You may need to modify this file to match the characteristics of your system in order for SPW to function properly.

You could make a batch file called WIN.BAT so that SPW will be loaded any time you type WIN to start Windows. Here are some ideas for you WIN.BAT, they assume WIN.COM is in the directory C:\WINDOWS and SPW31.EXE is in the directory C:\SPWIN...

This first one is the minimum neccessary to load SPW and start Windows:

```
        Rem  Do not display the commands in this BAT file as they execute.
@ECHO OFF

        Rem  Start SPW, which will invoke WIN.COM automatically.
C:\SPWIN\SPW31
```

This one uses a bootlog file and allows additional parameters to be passed to WIN.COM. The bootlog.txt file can be examined if an error occurs while trying to start Windows:

```
        Rem  Do not display the commands in this BAT file as they execute.
@ECHO OFF

        Rem  If there is not already a BOOTLOG.TXT file, then there is nothing to save.
IF NOT EXIST C:\WINDOWS\BOOTLOG.TXT GOTO START

        Rem  Append existing BOOTLOG.TXT to BOOTLOG.OLD (will be created if it doesn't already exist).
        Rem  Redirecting to NUL keeps any messages from displaying (i.e. 1 File copied).
COPY C:\WINDOWS\BOOTLOG.OLD+C:\WINDOWS\BOOTLOG.TXT C:\WINDOWS\BOOTLOG.OLD >> NUL

        Rem  Delete the previous BOOTLOG.TXT file. This way all information in BOOTLOG.TXT is from
        Rem  this instance of starting Windows.
DEL C:\WINDOWS\BOOTLOG.TXT

:START

        Rem  Start SPW, which will invoke WIN.COM automatically. The parameters will be passed to WIN.COM.
        Rem  /B means create a bootlog file. Typing C:\WINDOWS\WIN /? will list other parameters.
C:\SPWIN\SPW31 %1 %2 %3 %4 %5 %6 %7 %8 %9 /B
```

Activating SoftProbe/W

When SPW is active you can return to normal system processing by pressing **G** or **Q** and then **Enter**.

Upon loading SPWxx, SPW will become active just before it starts Windows. This provides the opportunity to monitor the Windows startup procedure if desired. When activated, you will see the SPW screen, which is described in the section *SPW's Screen*.

There are three ways that SPW is activated:

1) When it first loads it becomes active just before loading Windows.

2) When the hotkey is pressed. The default hotkey is **CTRL-SPACE**. This can be modified with the command SET HOTKEY=<scan code> from SPW itself (??) or in the SPWxx.INI file.

3) When a breakpoint is encountered. Breakpoints can occur on Reads, Writes, Execution, Interrupts, I/O and Faults and will be discussed in detail later.

Configuring SPW

SPW uses an INI file when it starts, **spw31.ini** or **spw95.ini**, which is expected to be in the same directory as the executable. If not present, default values are used. The INI file uses **[xxx]** to define sections and **#** to start a comment, which goes until the end of the line.

The three sections of the INI file are shown below as well as the definitions they can contain: (commands are explained in more detail in the *SPW Commands* section)

[SETUP]

LOG=n	# Reserves **n** KBytes for the Log Buffer. The default is 64k, the minimum is 8k. # Windows Event Messages are logged here.
CAP=n	# Reserves **n** KBytes for the Capture Buffer. The default is 40k, the minimum is ??. # I/O activity can be captured into this buffer.
SYM=n	# Reserves **n** KBytes for Symbolic Memory. The default is 40k, the minimum is ??. # This is used for ??
MEM=n	# Physical Memory on your system in MBytes. The default is ??, the minimum is ??.
NOFPU	# Use this only if your system does **not** have a Floating Point Unit. This setting is only # used by the SYS command and ??

[INIT]

SET HOTKEY=n # Sets SPW Hotkey to scan code **n** (decimal). Default is 57 (**^SPACE**).

SET SCR=VGA [nbrf] # SPW will use the primary VGA display with the colors **nbrf** (in hex). The default
is SET SCR=VGA 17 1F 74 13. See this command description and *Appendix A*.

SET SCR=MONO [nbrf] # SPW will use a Monochrome display with the colors **nbrf** (in hex). The default
is SET SCR=MONO 07 0F 70 07. See this command description and *Appendix A*.

SET SCR=LPTn # SPW will use remote display via LPT**n** (1..3).

SET SCR=COMn[:baud] # SPW will use remote display via COM**n**, n=1..4. **baud** is the baud rate (decimal) to
use on that port. The default baud is ??

SET PRN=LPT**n** # Use LPT**n** (1..3) as printer port. Using a non-existant port may cause your system to
crash. See this command description for additional options.

SET [~][^][@]key=cmd[;]# Assigns one or more commands to a keystroke combination. See this command
description for details.

SET CM**x**=[adr] # Set Code Marker **n** (0..9) to address **adr**. All default to 30:0 (selector 30, offset 0).

SET DM**x**=[adr] # Set Data Marker **n** (0..9) to address **adr**. All default to 28:0 (selector 28, offset 0).

These are not the only SPW commands that can be used used in the INI file, but they are the most common.

[SYM]

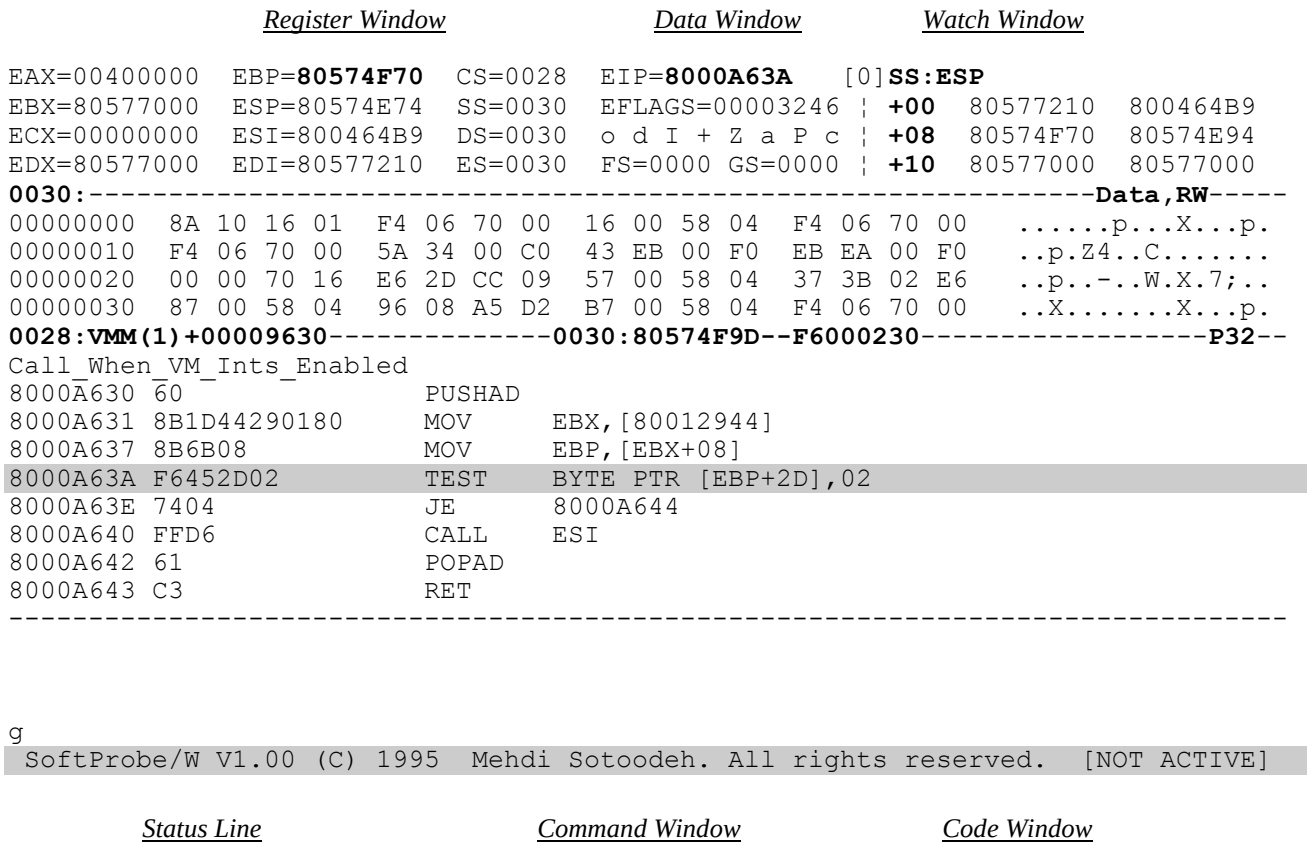
??

[END] # Indicates the end of the INI file.

SPW's Screen

This section describes the screen that is displayed when SPW is activated. Each part of this screen is identified and discussed, familiarity with the SPW screen will be helpful when reading the section *SPW Commands*.

A sample SPW screen is shown below with each major area of the layout identified. A more detailed discussion of each area follows.



SPW's Register Window

This window shows the General registers, Segment registers and Flags. The Control, Debug, Test and FPU registers can be seen by using the "SYS" command when SPW is active. Register and Flag values can be changed using the **R** command.

Flags and registers that are in bold (bright) text have changed. Keep in mind that the change(s) reflect the net result of **all** instructions executed between the last execution command (step, trace, go, etc.) and the current snapshot. Below the EFLAGS are symbolic representations for the Control and Status flags (except TF). Flags shown in lower case are clear (0) and flags shown in upper case are set (1).

```
EAX=00400000  EBP=80574F70  CS=0028  EIP=8000A63A
EBX=80577000  ESP=80574E74  SS=0030  EFLAGS=00003246
ECX=00000000  ESI=800464B9  DS=0030  o d I + Z a P c
EDX=80577000  EDI=80577210  ES=0030  FS=0000 GS=0000
```

The EFLAGS format is shown below, but detailed information is outside the scope of this document:

31..19	18	17	16	15	14	13..12	11	10	9	8	7	6	5	4	3	2	1	0
0..0	**AC**	**VM**	**RF**	**0**	**NT**	**IOPL**	**OF**	**DF**	**IF**	**TF**	**SF**	**ZF**	**0**	**AF**	**0**	**PF**	**1**	**CF**

Note 1: a 1 or 0 indicates a reserved bit in EFLAGS.
Note 2: bit 18 is 0 for CPUs below 80486.

System Flags

AC	Alignment Check
VM	Virtual 8086 Mode
RF	Resume flag
NT	Nested Task flag
IOPL	I/O Privilege Level
IF	Interrupt Enable flag

Status Flags

OF	Overflow flag
CF	Carry flag
TF	Trap flag
SF	Sign flag
ZF	Zero flag
AF	Auxiliary Carry flag
PF	Parity flag

Control Flags

DF Direction flag

SPW's Watch Window

This window displays a portion of memory in several formats, controllable by you: bytes (8 bits), words (16 bits), dwords (32 bits) or ASCII (8 bits). There are 8 possible **watches**, each of which may be an address, a register pair or a combination of the two. SPW remembers all 8 watches, but only 1 can be displayed at a time. See the **Wx** command.

You can not manually scroll this window; however, it is automatically adjusted so that the upper left corner always shows the contents of the current watch expression. For example if the watch expression involves a register and this register changes, the address and data of the watch window will reflect this.

At the upper left of this window you will see the current watch expression and the current watch number. The current watch number shows which of the watches (0..7) is currently displayed. When SPW is loaded watch 0 will be displayed and the watches will have the following values:

```
[0]= SS:ESP    [1]= DS:ESI    [2]= ES:EDI    [3]= DS:EDX
[4]= ES:EBX    [5]= SS:EBP    [6]= CS:EIP    [7]= DS:0
```

```
watch
number      watch expression
↓       ↙
[0]SS:ESP
   +00  80577210  800464B9
   +08  80574F70  80574E94
   +10  80577000  80577000
    ↑              ↑
offset from        data at watch expression
watch expression   address (24 bytes total).
address
```

Note that the "offset from watch expression address" is only shown if the data is displayed as double words (dwords). See the commands **W, WB, WW, WD** and **WA**.

The Data Window

This window is similar to the Watch Window, but you have more control over it. The window is scrollable and you can set and select upto 10 markers. The data can be displayed in the same formats as the Watch Window (ASCII, byte, word, dword).

```
      selector of                                                  selector
      displayed data                                 selector type attributes
   ↙                                                           ↘    ↓
0030:-------------------------------------------------------------Data,RW-----
00000000  8A 10 16 01  F4 06 70 00  16 00 58 04  F4 06 70 00   ......p...X...p.
00000010  F4 06 70 00  5A 34 00 C0  43 EB 00 F0  EB EA 00 F0   ..p.Z4..C.......
00000020  00 00 70 16  E6 2D CC 09  57 00 58 04  37 3B 02 E6   ..p..-..W.X.7;..
00000030  87 00 58 04  96 08 A5 D2  B7 00 58 04  F4 06 70 00   ..X.......X...p.
  ↑                          ↑                                      ↑
offset              hex data at selctor:offset                ascii data at
                                                              selector:offset
```

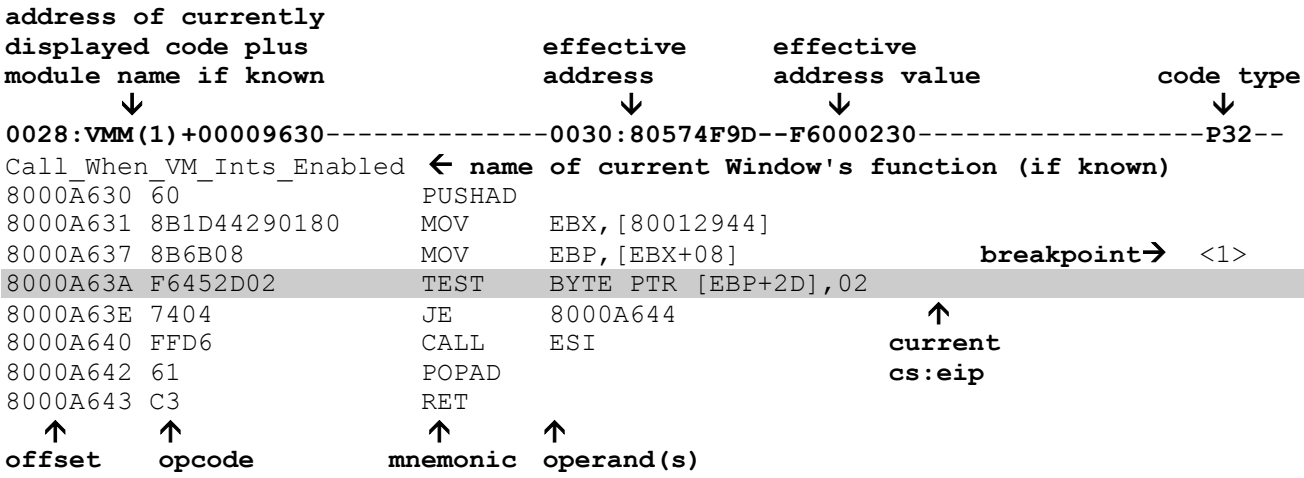
```
Selector types can be:
  Code, Data, LDT, 386TSS, <V86>, ??
Selector attributes can be:
  RW = Read/Write, RO = Read Only, XR = Read/Execute, XO = Execute Only ??
  B = ??
```

The Code Window

The Code window displays a section of code, normally at the current CS:EIP, but you may view another area by scrolling, specifying another address or selecting a Code Marker. The row shown in highlighted (reverse video) text is the instruction at the current CS:EIP.

The opcode column shows the opcode for a given instruction upto 8 bytes. If an opcode is longer than this you can view it by copying it to the Data window using the **D** command.

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



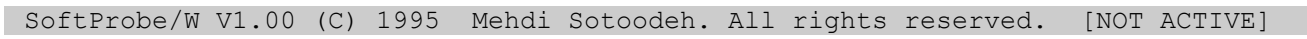
In this example we are currently executing code in the module **VMM** (Virtual Machine Manager) and we are in the Window's function **Call_When_VM_Ints_Enabled.** These are not always known to SPW and are not displayed in those cases.

The current code is **P32**, 32-bit protected mode. Possible code types are: ??

The Status Line

The Status Line is the bottom row the SPW screen and shows whether or not SPW is active. The inactive status line is useful when using a 2nd monitor, since the SPW screen might be shown on this monitor even when SPW is not active. When active this line is also used to input commands to SPW.

An inactive status will look similar to this:



An active status will look similar to this:

The Command Window

This area between the Code Window and the Status Line shows the last few SPW commands entered. It is also used to display messages, such as "Invalid Command" and "Break Due To Hot Key".

SPW Commands

Commands are entered either from **spwxx.ini** when SPW loads or on the Status Line when SPW is active. In order to make the command descriptions simpler and shorter a number of abbreviations and terms are used, these are defined in *Appendix C*. You may wish to review this appendix now. Keep in mind that **all number are in hex** unless otherwise specified.

Breakpoints

Breakpoints are used to halt system execution and activate SPW in order to inspect or modify Registers, Code, Data or SPW functionality. You have several breakpoint options to choose from, which cover most possible scenarios.

There are some basic points which are worth mentioning here instead of duplicating them in several command descriptions. When a breakpoint is set, it has a number assigned to it, starting at 1 and increasing in sequence. However, if any gaps occur (see **BC** below) those numbers will be reused first (the lowest available number is used). These numbers are used by some SPW commands to reference one or more breakpoints. They will also be displayed in the SPW Code Window as **<n>** when a breakpoint address is visible in that window.

BP [adr [cond]]

Sets a breakpoint at the specified address and with the specified condition. A break will occur **before** executing the instruction at this address **if** the condition is true. The default address and condition are: the current CS:EIP and N=1 (break every time). You may set up to ?? breakpoints using **BP**.

This is a software breakpoint and SPW may need to monitor the execution process to see if the instruction at a breakpoint is about to be executed. This can slow the system a little, but is normally not a problem. If system performance is an issue or if you are trying to set a breakpoint in ROM use the **BPX** command instead.

BP (same as BP CS:EIP N=1)	Set BP at current CS:EIP and break every time.
BP N=3 (same as BP CS:EIP N=3)	Set BP at current CS:EIP and break every 3rd time.
BP DS:ESI+EAX+5 N=4 EAX+EBX > = 10000	Set BP at the address given by DS:ESI+EAX+5. Break every 4th time **if** EAX+EBX >= 10000 (the 4th time).

BPX [adr [cond]]

Same as **BP** except this is a hardware breakpoint (see *Appendix B*). This means that SPW does **not** modify the instruction where the breakpoint is being set and that system performance is **not** slowed by the debugging process. This command allows breakpoints to be set in ROM and is useful with real-time applications. The CPU supports 4 hardware breakpoints.

BPB adr [cond]

This hardware break (see *Appendix B*) occurs when the **byte** at the given address is **read** or **written** and the condition is true. Otherwise it is the same as **BPX**.

BPW adr [cond]

This hardware break (see *Appendix B*) occurs when the **word** at the given address is **read** or **written** and the condition is true. Otherwise it is the same as **BPX**.

BPD adr [cond]

This hardware break (see *Appendix B*) occurs when the **dword** at the given address is **read** or **written** and the condition is true. Otherwise it is the same as **BPX**.

BPBW adr [cond]

This hardware break (see *Appendix B*) occurs when the **byte** at the given address is **written** (no break on a read) and the condition is true. Otherwise it is the same as **BPX**.

BPWW adr [cond]

This hardware break (see *Appendix B*) occurs when the **word** at the given address is **written** (no break on a read) and the condition is true. Otherwise it is the same as **BPX**.

BPDW adr [cond]

This hardware break (see *Appendix B*) occurs when the **dword** at the given address is **written** (no break on a read) and the condition is true. Otherwise it is the same as **BPX**.

BM module

This is a software break that will occur whenever the specified module is reloaded.
?? (BM works with modules especially with WIN32. Use MOD32 to list
them, then something like BM KERNEL32 will stop whenever KERNEL32 is
reloaded due to the linkage of any app that uses it (every win32).)

BD list | *

Disables one or more breakpoints, but does not delete them (SPW still remembers their addresses and conditions). This command works on any breakpoint that has a breakpoint number (breakpoints set with any SPW breakpoint command **except BI1, BI3,** or **BF**). Disabling an already disabled breakpoint leaves that breakpoint disabled. Disabled breakpoints can be re-enabled with **BE**.

List is 1 or more breakpoint numbers separated by a white space.

BD	No effect ('*' or breakpoint list must be given).
BD*	Disables all breakpoints.
BD 3	Disable breakpoint 3, if it exists.
BD 1 3 4	Disable breakpoints 1, 3 and 4 if they exist. Breakpoint 2 is not affected. The command aborts when an invalid breakpoint number is encountered. Using this example, suppose that only breakpoints 1 and 4 exist. Breakpoint 1 will be disabled and since breakpoint 3 does not exist processing stops. Breakpoint 4 will not be affected.

BE list|*

Enables one or more breakpoints that were disabled with the **BD** command. Enabling an already enabled breakpoint leaves that breakpoint enabled.

BE	No effect ('*' or breakpoint list must be given).
BE*	Enables all breakpoints.
BE 3	Enables breakpoint 3, if it exists.
BE 1 3 4	Enables breakpoints 1, 3 and 4 if they exist. Breakpoint 2 is not affected. The command aborts when an invalid breakpoint number is encountered. Using this example, suppose that only breakpoints 1 and 4 exist. Breakpoint 1 will be enabled and since breakpoint 3 does not exist processing stops. Breakpoint 4 will not be affected.

BC list

Clears (deletes) one or more breakpoints. Cleared breakpoints can **not** be re-enabled.This

removes the breakpoint from SPW's memory and the associated breakpoint numbers become available for use and will not be displayed by the **BL** command. Note: breakpoint numbers are not shifted to close gaps. If you have two breakpoints (1 and 2) and delete breakpoint 1, breakpoint 2 will stay number 2 and number 1 will be assigned to the next breakpoint that is set.

BC	No effect ('*' or breakpoint list must be given).
BC*	Clears all breakpoints.
BC 3	Clears breakpoint 3, if it exists.
BC 1 3 4	Clears breakpoints 1, 3 and 4 if they exist. Breakpoint 2 is not affected (it stays number 2). The command aborts when an invalid breakpoint number is encountered. Using this example, suppose that only breakpoints 1 and 4 exist. Breakpoint 1 will be cleared and since breakpoint 3 does not exist processing stops. Breakpoint 4 will not be affected.

In some case a breakpoint may become inaccessable, if this happens a message will displayed and you will be prompted as to whether or not you wish to clear the breakpoint anyway. ??

BL

Lists all breakpoints and their assigned breakpoint numbers in a scrollable window. Unassigned breakpoint numbers are not shown. The following information is displayed for each breakpoint:

No = Breakpoint number.
Type = Breakpoint type (**BP,BPX,BPB,BPW,BPD,BPBW,BPWW,BPDW,BM**).
Address = Breakpoint address, selector/segment (16-bit) and offset (32-bit).
Enabled = Breakpoint enabled, Yes or No. A "+" indicates the address is not accessable.
Count = Breakpoint occurs every **n**th time and only if the Condition is true.
Condition = Breakpoint occurs if the displayed condition is true (and count is true). A blank condition means always break if count is true.

BIO p [cond]

This software break occurs if I/O port **p** is **about** to be read or written **and** the condition is true. This command works only on non-priviledged processes (running in rings 1,2 or 3).

BIO 3BC	Break on instructions that access port 3BC, every time.
BIO 42 EAX=8	Break on instructions that access port 42 **if** EAX contains the value 8.

BPIO p [cond]

This is the same as **BIO**, except that it is a hardware break and will also work on priviledged processes (ring 0). **This command can be used only on Pentiums and above.**

BI n [cond]

This software break occurs if INT**n** is **about** to occur **and** the condition is true. Interrupts 1 and 3 are special, see the **BI1** and **BI3** commands.

BI 21	Break on INT 21 instructions, every time.
BI 16 N=2 EBX < 5	Break on INT 16 instruction, every other time **if** EBX < 5.

BI1 [ON|OFF]

This is a general SPW setting and is **not** assigned a breakpoint number. This setting controls whether or not SPW breaks **after** an INT 1 (single step) has occured. The default is ON.

BI1 ON	Break when an INT 1 occurs.
BI1 OFF	Do not break when an INT 1 occurs.
BI1	Display the current status, ON or OFF.

BI3 [ON|OFF]

This is a general SPW setting and is **not** assigned a breakpoint number. This setting controls whether or not SPW breaks **after** an INT 3 (debug break) has occured. The default is ON.

BI3 ON	Break when an INT 3 occurs.
BI3 OFF	Do not break when an INT 3 occurs.
BI3	Display the current status, ON or OFF.

BF [GP|PF|SF|OP] [ON|OFF]

This is a general SPW setting and is **not** assigned a breakpoint number. This setting controls whether or not SPW breaks on **GP** (General Protection Faults), **PF** (Page Faults), **OP** (Invalid Opcodes) and **SF** (Stack Faults). All these settings are ON by default. When one of these faults is trapped you will see a message indicating which fault caused the break.

BF	Display current settings for **GP, PF, OP** and **SF** .
BF GP OFF	Disable breaks on General Protection Faults.
BF SF ON	Enable breaks on Stack Faults.

ZAP

Removes an embeded INT 3 or INT 1 instruction. Some applications may contain these instructions, which may cause a break to SPW. An INT 3 is replaced by one NOP (no operation instruction) and an INT 1 is replaced by two NOPs.

This instruction will only work when the current CS:EIP is on the instruction **after** an INT 1 or INT 3, for example after a **BI1** or **BI3** break. You can also manually change CS:EIP to an instruction following an INT 1 or INT 3. Only the memory image is modified, the original disk image is not affected.

Execution
The instructions in this section control execution of code.

T n

Traces **n** instructions, the default is 1. During this execution SPW will run in Trace Mode (single step mode), which is relatively slow. The trace command will "go into" every Call and execute every cycle of a Loop instruction. If you do not want to trace into a Call or Loop use the **P** command, set a breakpoint after those instructions or use the **G adr** command..

If **n** is large it may take a while for this command to complete. The data you entered on SPW's Status Line will remain until the command is completed. This lets you know that the command is still in progress. SPW's hotkey can be used to abort this command if you specified a large **n**.

T	Trace 1 instruction.
T 100	Trace 100 instructions.
T FFFFFFFF	Trace FFFFFFFF instructions. This **will** take some time!

Entering **T** while CS:EIP is at the CALL statement will execute one instruction and CS:EIP will be at CS:800852D4, not at the CMP instruction. Compare this with to the **P** command.

```
80085168 E867010000          CALL 800852D4
8008516D3B1568640880         CMP  EDX,[80086468]
```

P

The Step command executes 1 "line" of code, which is normally 1 instruction except as follows:

1) CALL
2) INT
3) LOOP, LOOPE, LOOPZ, LOOPNE, LOOPNZ
4) REP, REPE, REPZ, REPNE, REPNZ
5) Priviledged commands when running in non-priviledged mode:
   CLI, STI, IN, OUT, CLTS, HLT, LGDT, LIDT, LLDT, LMSW, LTR and MOVs to system registers.

Because **P** sets a temporary breakpoint after the current instruction, the above cases may cause many instructions to be executed even though the end result is that CS:EIP ends up at the following instruction. For example using **P** while on a Call instruction will execute all the instructions in the called routine and then stop after returning to the instruction after the Call instruction. For example, entering **P** when CS:EIP is at the CALL will execute code until CS:EIP is at the CMP statement. Compare this with to the **T** command.

```
80085168 E867010000          CALL 800852D4
8008516D3B1568640880         CMP  EDX,[80086468]
```

In the unlikely case that control is not returned to the following instruction then the **P** command ends up acting like a **G** command. This can happen if a called routine changes it return address. In this case the breakpoint is not removed and will show up when the **BL** command is executed or if that instruction is displayed in the Code Window. You can remove or disable this breakpoint with **BC** or **BD**.

If you are running in non-priviledged mode (rings 1,2 or 3) this command will step over a priviledged instruction, whereas the **T** command will trace into the handler for that instruction.

G/Q

This exits SPW and returns to the host for normal processing. Normal processing will continue until SPW's hotkey is pressed or until a breakpoint or fault is encountered. You can think of this command as meaning either Go or Quit.

G	Return to normal processing.
Q	Return to normal processing.

G adr

Same as **G** except a temporary breakpoint is set at the specified address.

G CS:80001234	Return to normal processing and break when address CS:80001234 is encountered. SPW might be reactivated before this breakpoint is reached if another break, fault or hotkey occurs.

G = adr

This command first sets CS:EIP to the specified address and then performs a **G** command. This command is used to restart execution from a certain address. This command has no effect if the current CS:EIP is already at the specified address.

This command can be dangerous, because you might skip code that needs to be executed. For example if items have been pushed onto the stack and you skip over the code that removes these items, you could crash your system.

GC

Similar to **G addr**, but normal processing occurs until Client Code is encountered. This is the client that was interrupted when SPW was activated via the hotkey. If SPW was activated because of a breakpoint then this command will have no effect, because CS:EIP is already in the client code. Also using **GC** when CS:EIP is already in client code will have no effect (i.e. using **GC** twice in a row).

GF

Same as **G** except faults are ingored. SPW is only activated by the hotkey or a breakpoint. This temporarily turns off the breakflags as if the command **BF OFF** was given. However, once SPW is reactivated the breakflags revert to their previous values.

END TASK [errcode]

Terminates the current task (if possible) and issues the error code **errcode**. The default error code is 0. ??

END VM

Terminates the current Virtual Machine and returns to its parent process. If a DOS Box is the current VM, then the DOS Box is terminated and control is returned to Windows. If Windows is the current VM, then Windows is terminated.

RESET

This will cause your system to reboot! You will **not** be asked to confirm this action!

IF exp cmd

Used to conditionally execute one command. Interpret it as "if **exp** is true then execute **cmd**".

IF DS:1A45 < 5 T 3 Trace 3 instructions **if** the memory value at DS:1A45 is < 5.

WHILE [exp cmd[;cmd..]]

An expanded for of the **IF** command that can be used to conditionally execute one or more commands multiple times. It means "while **exp** is true execute **cmd**(s)". This command can be interrupted by pressing a key.

WHILE	Repeat the previously defined **WHILE** (if there is one).
WHILE CS=28 P	Step through code while CS=28 (stop when CS changes).
WHILE EAX<5 G 9B23; T; R EAX EAX+1	While EAX < 5 we will execute until CS:9B23, Trace 1 instruction and then increment EAX. Without the **T** the next **G 9B23** command will not have any effect.

Code and Data

The commands in this section are primarily for controlling the display of data (memory). These commands affect the Data Window, Watch Window and Code Window. Other functions allow manipulation of memory in various ways.

In cases where an **adr** or **blk** is used, the default segment is the one currently being shown in the Data Window. Remember that all values are entered and displayed as hex.

D [adr]

Displays the data at **adr** in the Data Window using the current data format (**DB, DW, DD, DA, DL, DP**). If **adr** is not given, then the command acts like a Page Down function.

D	Display the next page of data (higher address).
D GS:EIP+24	Display data using GS for the selector (segment) and EIP+24 as the offset.

DB [adr]

Display data at the given **adr** as bytes. If **adr** is not given then the current data in the Data Window is redisplayed using byte formatting.

DW [adr]

Same as **DB**, but displays data as words.

DD [adr]

Same as **DB**, but displays data as dwords.

DA [adr]

Same as **DB**, but displays data as ASCII.

DL [adr]

Displays data in a scrollable window. The data is always displayed as bytes. The default value of **adr** is DS:0. This command allows you to scroll through all available Linear Memory.

DP exp

Displays physical memory in a scrollable window. The data is always displayed as bytes. The default value of **adr** is 0:0. The maximum value depends upon ?? This command allows you to scroll through the entire Physical Memory space.

W [n]

Displays Watch **n** (0..7) in the Watch Window.

W	Displays the next Watch. If the current Watch is 7 then display Watch 0.
W 3	Displays Watch 3.
W EAX	Uses the value of EAX for **n**. If EAX > 7 an error will be generated.

WB [exp]

Redefines the current Watch to the given **exp** and displays data as bytes. If **exp** is not given, then only the format of the current Watch is affected.

WB	Changes the format of the current Watch to bytes, does not affect the Watch address.
WB SS:EIP	Sets the current Watch to show data at SS:EIP as bytes.

WW [exp]

Same as **WB**, but data is displayed as words.

WD [exp]

Same as **WB**, but data is displayed as dwords.

WA [exp]

Same as **WB**, but data is displayed as ASCII.

M blk adr

Moves (copies) data from the specified **blk** to the memory starting at **adr**.

M CS:10 4F DS:110	Move CS:10..CS:4F to DS:110..DS:14F (40 bytes).
M CS:10 L 40 DS:110	Same as above using the alternate form for **blk**.
M 5 F 30	Move x:5..x:F to x:30..x:3A (B bytes, x = segment shown in Data Window).

C blk adr

Compares the memory specified by **blk** with memory starting at **adr**. If the data in the two memory blocks is identical nothing will be displayed (it will appear that no action has occured). However, if the memory is different a scrollable window will be displayed showing the addresses, hex values and ASCII values of the bytes that differ.

C CS:10 4F DS:110	Compare CS:10..CS:4F with DS:110..DS:14F (40 bytes).
C CS:10 L 40 DS:110	Same as above using the alternate form for **blk**.
C GS:5 L CL+20 30	Compare GS:5..GS:5+CL+20-1 with GS:30..GS:30+CL+20-1 (CL+20 bytes).

F blk str

Fills the memory specified by **blk** with the data given in **str**. If an expression that results in a numeric value is given for **str**, only the value of the low byte is used.

F ES:123 L 50 '0'	Fills ES:123..ES:172 with the character '0' (30 hex).
F ES:123 172 30	Same as above using alternate forms for **blk** and **str**.
F ES:123 172 124530	Same as above. Only the low byte of 124530 is used.
F GS:0 5 'ZAPHOD'	Fills GS:0..GS:5 with the characters 'ZAPHOD'. The same result can be obtained by using E GS:0 'ZAPHOD'.

E adr str

Enters the data in **str** into the memory starting at **adr**. If an expression that results in a numeric value is given for **str**, only the value of the low byte is used. The main difference between this command and the **F** command is that you do not specify an explicit size, it's based on **str**.

E 50:123 '123'	Puts the characters '123' into 50:123..50:125.
E 50:123 31 32 33	Same as above using alternate forms for **blk** and **str**.
E CS:A613 FFFFFF30	Puts a 30 ('0') into CS:A613. Only the low byte of FFFFFF30 is used.
E GS:0 'ZAPHOD'	Puts the characters 'ZAPHOD' into GS:0..GS:5. The same result can be obtained by using F GS:0 5 'ZAPHOD'.

S [blk str]

Search for **str** in memory block **blk**. If no parameters are given the search continues from the current position instead of from the start of the block (find next). ??

S	Find next.
S DS:0 L FFFF 'Volume'	Searches for 'Volume' in DS:0..DS:FFFF.

PC blk

Prints the data in **blk** using a format similar to the Code Window display. The current printer

port is used (specified by **SET PRN**) and printing can be aborted by pressing ESC. The printing always begins at the starting address, but may continue a few bytes past the ending address in order to print a complete instruction..

PC CS:EIP-10 L 30	Prints (as code) the memory at CS:EIP-10..CS:EIP+1F.

PD blk

Prints the data in **blk** using a format similar to the Data Window display. The current printer port is used (specified by **SET PRN**) and printing can by pressing ESC. The printing always begins at the starting address, but may continue a few bytes past the ending address in order to print a complete row of 16 bytes. The data is always shown as bytes.

PD GS:8000A01B L 100	Prints (as data) the memory at GS:8000A01B..GS:8000A11A.

SET CMn[[=]adr]

Set Code Marker **n** (0..9) to the specified **adr**. If **adr** is not given then the marker is set to the address currently at the top line of the Code Window. These are bookmarks that allow you to mark a location for later reference (using **CMn**). The address specified does not have to be valid code. The default for all Code Markers is 28:0 (selector 28, offset 0).

SET CM7	Sets Code Marker 7 to the address at the top of the Code Window.
SET CM3 = EAX+EBX:4	Sets Code Marker 3 to segment EAX+EBX, offset 4.

SET DMn[[=]adr]

Set Data Marker **n** (0..9) to the specified **adr**. If **adr** is not given then the marker is set to the address currently at the top line of the Data Window. These are bookmarks that allow you to mark a location for later reference (using **DMn**). The default for all Data Markers is 30:0 (selector 30, offset 0).

SET DM7	Sets Data Marker 7 to the address at the top of the Data Window.
SET DM3 = GS+10:EBX	Sets Data Marker 3 to segment GS+10, offset EBX.

CMn

Goto Code Marker **n** (0..9). Changes the address displayed in the Code Window to that of the specified Code Marker. If the Code Marker was not set (using **SET CMn**) the default is used (28:0, selector 28, offset 0).

CM6	Goto Code Marker 6.

DMn

Goto Data Marker **n** (0..9). Changes the address displayed in the Data Window to that of the specified Data Marker. If the Data Marker was not set (using **SET DMn**) the default is used (30:0, selector 30, offset 0).

DM6	Goto Data Marker 6.

MC n

Scroll the Code Window **n** lines (-FFFFFFFF..FFFFFFFF). Positive values result in higher numbered addresses, negative values result in lower numbered addresses. Large values of **n** may take a long time to process because the start and end of instructions must be determined.

MC -2	Scroll "back" 2 lines of code (smaller address).
MC 1	Scroll "ahead" 1 line of code (larger address).

MD n

Scroll the Data Window **n** screens (-FFFFFFFF..FFFFFFFF). Positive values result in higher numbered addresses, negative values result in lower numbered addresses.

MD -2	Scroll "back" 2 screens of data (smaller address).
MD 1	Scroll "ahead" 1 screen of data (larger address).

Port I/O

This section contains commands that deal with input (reads) from and output (writes) to ports.

I port

Read a byte from I/O port **port** (0..FFFF). The result is displayed in the Command Window.

I 3BC	Read a byte from port 3BC.
I EDI	Read a byte from the port specified in the **low word** of EDI.

IW port

Read a word from I/O port **port** (0..FFFF). The result is displayed in the Command Window.

ID port

Read a dword from I/O port **port** (0..FFFF). The result is displayed in the Command Window.

O port val

Write the byte **val** (0..FF) to I/O port **port** (0..FFFF).

O 3BC AB	Write the hex value AB to port 3BC.
O EDI EAX	Write the **low byte** of EAX (AL) to the port given in the **low word** of EDI.

OW port val

Write the word **val** (0..FFFF) to I/O port **port** (0..FFFF).

OD port val

Write the dword **val** (0..FFFFFFFF) to I/O port **port** (0..FFFF).

CAP [START | STOP | port1 [port2 [port3]]]

This command is used to capture and display I/O activity on up to 3 ports. It only works for applications running in non-priviledged modes (rings 1,2 or 3).

CAP	Displays the data in the Capture buffer (does not clear the buffer).
CAP 60	Clears the buffer and begins capturing I/O on port 60.
CAP 278 378 3BC	Clears the buffer and begins capturing I/O on ports 278, 378 and 3BC.
CAP START	Clears the buffer and begins capturing I/O on the previously defined port(s).
CAP STOP	Stops capturing (does not clear the buffer).

BIO p [cond]

See this same command in the *Breakpoints* section of this document.

BPIO p [cond]

See this same command in the *Breakpoints* section of this document.

System Information

This group of commands allows you to view information about various system structures. There are no commands that let you change these structures.

IDT [n]

Displays Interrupt Descriptor Table (IDT) entries starting with interrupt **n** (0..FF), the default is the lowest used interrupt number. There is only one IDT for the entire system and its physical address is kept in the IDTR (IDT Register). The IDTR specifies the 32-bit base address (physical) and the 16-bit Limit of the IDT. Adding the Limit to the Base gives you the last byte of the IDT.

Each IDT entry is 64-bits (8 bytes) and contains a Gate Descriptor for a particular interrupt. Gates allow execution to transfer from code at a lower priviledge level to code at a higher priviledge level in a secure manner. This happens when an interrupt occurs, because the Interrupt Handler may be in ring 0 while the interrupted process may be running in ring 3. A more detailed explanation of Gates is beyond the scope of this document.

When an interrupt occurs the proper IDT entry is found by: IDT base + (8*Int number). This entry specifies the address of the Interrupt Handler and other information. SPW shows the following for each IDT entry that is in use:

Int#	Interrupt descriptor number.
P	**P** means the segment is present in memory, **NP** means not present.
DPL	Descriptor Protection Level, 0..3. 0 is the highest.
Type	One of: 286TaskGate, 286IntGate, 286TrapGate, 386TaskGate, 386IntGate, or 386TrapGate (Call Gates are not valid IDT types).
Address	Selector (segment) and offset of the interrupt handler.

GDT [sel]

Displays Global Descriptor Table (GDT) entries starting with selector **sel**, the default is the lowest used selector. There is only one GDT for the entire system and its physical address is kept in the GDTR (GDT Register). The GDTR specifies the 32-bit base address (physical) and the 16-bit Limit of the GDT. Adding the Limit to the Base gives you the last byte of the GDT.

Each GDT entry is 64-bits (8 bytes) and contains a Descriptor. These descriptors reference tasks or other operating system items that are global. Each individual task will also have its own LDT. SPW shows the following for each GDT entry that is in use:

Selector	Selector number.
Base	The starting offset within segment.
Limit	Size of selector in bytes.
DPL	Descriptor Priviledge Level (0..3).
Size	16-bit or 32-bit offsets.
P	**P** means the segment is present in memory, **NP** means not present.
Type	Code, Data, LDT, 386TSS, NOT IN USE and the same types as IDT.
Attributes	`RW = Read/Write, RO = Read Only, XR = Read/Execute, XO = Execute Only, B = ??`

LDT [sel]

Displays Local Descriptor Table (LDT) entries starting with selector **sel**, the default is the lowest used selector. There is an LDT for each task.

Each LDT entry is 64-bits (8 bytes) and contains a Descriptor. These descriptors reference local tasks. SPW shows the following for each LDT entry that is in use:

Selector	Selector number.
Base	The starting offset within segment.
Limit	Size of selector in bytes.
DPL	Descriptor Priviledge Level (0..3).
Size	16-bit or 32-bit offsets.
P	**P** means the segment is present in memory, **NP** means not present.
Type	Code, Data, NOT IN USE and the same types as IDT.
Attributes	`RW = Read/Write, RO = Read Only, XR = Read/Execute, XO = Execute Only, B = ??`

SYS

Displays the following system registers and related information:

GDTR	Holds the linear Base address and Limit of the GDT, displayed as: GDT:BASE=nnnnnnnn LIMIT=nnnn.
IDTR	Holds the linear Base address and a Limit of the IDT, displayed as: IDT:BASE=nnnnnnnn LIMIT=nnnn.
LDTR	Holds the selector of the current LDT descriptor, displayed as LDT=nnnn.
TR	Holds the selector of the TSS of the current process, displayed as TSS=nnnn.
DRn	Debug registers: DR0..DR3, DR6, DR7 (DR4 and DR5 are not available).
CRn	Control registers: CR0, CR2, CR3 (CR1 is not available).
CS:EIP	Contents of FCS:FIP (the CS:EIP value when the FPU was invoked).
Opcode	The FOP register. This is the opcode of the instruction that invoked the FPU.
Operand	Contents of FOS:FOO. The address of the last memory operand (if any).
Status	FPU Status Word Register (see *Appendix D*).
Control	FPU Control Word Register (see *Appendix D*).
Tag	FPU Tag Word Register (see *Appendix D*).
ST(n)	These 8 registers are used for calculations (see *Appendix D*).

TSS [sel]

Displays the TSS for selector **sel**. The TSS is used to save the context of a task in a multitasking environment. SPW displayes the following TSS information:

Regs	All normal registers including segment registers and EFLAGS.
SSn:ESPn	Stack pointers for each ring, **n** = 0..3.
CR3 (PDBR)	Physical address of the current page directory for this task.
LDTR	Contains the selector of this task's LDT.
Back Link	The selector of the previously executing task (if any).
Trapped I/O Ports	A map of any I/O ports that this task traps (x = trapped).

VXD [vxdname]

Displays information about the VxD **vxdname**. If no **vxdname** is given, then all VxDs are shown in a scrollable window. The following information is displayed:

Name	The 8 character VxD name.
Vers	The Windows version associated with that VxD: 3.0, 3.1, ??.
DevID	??
CtrlProc	The address of the VxD Control Procedure.
PM API	The address of the Protected Mode API entry point.
V86 API	The address of the V86 (Real) Mode API entry point.
Ref Data	The address of the Reference Data ??
DDB	??
InitOrder	?? (only shown if **vxdname** is given).
#Service	Shows the number of service routines in this VxD. If **vxdname** was given this is replaced by the actual addresses and names (if known) of the service routines.

PAGE [adr]

Displays Page Information in a scrollable window starting at address **adr**. Pages are used to convert linear addresses into physical addresses. A 32-bit linear address is split up as follows:

```
nnnnnnnnnn nnnnnnnnnn nnnnnnnnnnnn
(PDE index)   (PTE index)   (physical offset)
```

The high 10 bits of the linear address index into the current Page Directory. The current Page Directory is determined by the contents of CR3 (PDBR). Each of 1024 (2^{10}) entries in the Page Directory is called a Page Directory Entry (PDE) and points to a Page Table.

The middle 10 bits of the linear address index into the Page Table identified by the PDE. Each Page Table has 1024 (2^{10}) entries called Page Table Entries (PTE). Each PTE contains a 32-

bit value. The high 20 bits are the base of the pysical address and the low 12 bits are status bits.

The low 12 bits of the linear address are combined with the base of the physical address (obtained from the PTE) to create a the desired 32-bit physical address. The physical offset has a range of 4096 (2^{12}) bytes.

The **PAGE** command shows the following information:

Directory Base	The physical memory address of the Page Directory Base, determined by CR3.
Linear Address	The linear address range that has the same PDE and PTE indexes. This is a 4096 byte range since the physical offset (low 12 bits) is not important.
PDE	The PDE value (Page Table's physical address).
PTE	The PTE value (physical base and status), 0 means the Page is not present.
Physical	The physical base address taken from the PTE (status bits removed).
Attributes (of this page)	**D** = Dirty. A write has occured in this page since it was loaded. If a page is not dirty, it can be discarded without updating the disk image, which saves time.
	A = Accessed. The page has been read or written. Shows which pages have been accessed since loading or since the O.S. last cleared this bit. Used to determine which pages to keep when freeing memory.
	PCD = Page Cache Disable. Disables caching and write-through for this page. This is only available on 486 processors and above.
	PWT = Page Write Through. Used by external cache only, not by the processor. This is only available on 486 processors and above.
	U/S = User / Supervisor. Rings 0,1,2 are considered Supervisor, ring 3 is User.
	R/W = Read only / Writable. The effect of this bit depends on **U/S** and CR0's Write Protect (WP) bit. WP is only on 486 processors and above.
	P = Present. Tells whether or not this page is in memory.

PDE [adr]

Displays Page Directory Entries (PDE) in a scrollable window starting at address **adr**. Read the description of the **PAGE** command if you need more information. **PDE** displays the following:

Directory Base	The physical memory address of the Page Directory Base, determined by CR3.
Linear Address	The linear address range that has the same PDE indexes. This is a 4,194,304 (2^{22}) byte range since the PTE index (middle 10 bits) and the physical offset (low 12 bits) are not important. See the **PAGE** command.
PageTable	This is the physical memory address of the Page Table to use.
Attributes	Same as the **PAGE** command.

HEAP

Display global heap ??

Hndl	Handle
Base	Base address
Length	
#sel	
Owner	

TASK

Displays the current tasks. Windows itself shows up as a task called PROGMAN. ??

TaskName	This is the name of the task.
Sz	
TaskDB	
hModule	
hInst	
hQueue	
PSP	
Parent	

ST

Displays the call stack. The most recent call is at the beginning of the list. Not all calls will show up here ??

SS:ESP	The top value (above the frame) is the current value of SS:ESP. The other values (below the frame) are for SS:ESP when that particular Call was made. The return address for this call is at this location on the stack.
Caller CS:EIP	The address of this Call instruction.
Callee CS:EIP	The address of the code that was called by this Call instruction.
Owner	The owner (if known) of the **called** code. This is the code at **Callee CS:EIP**.

VM

Displays list of currently running VMs (virtual machines) ??

VM Handle
VM ID
Status
High Lin
Client

MOD [sel|hMod|name]

Lists or displays 16-bit modules that are currently loaded. If no parameter is given all currently loaded modules are displayed in a scrollable window with the following information:

hMod	The module's selector (handle).
Module	The module name in memory.
Filename	The path and file name that this module was loaded from.

If you specify a module selector (**sel** or **hMod**) or a module name you will get the following information about that module, if it exists:

ModuleDB	The module's selector (same as hMod above).
Module Name	The module name in memory (same as Module above).
Expected Win Ver	The minimum Windows version required.
Flags	??
Filename	The path and file name that this module was loaded from.
DGROUP	The data segment of the module.
Stack	??
Heap	??
Usage	??
Seg#	The ordinal number assigned to a selector within this module, if any.
Sel/Hnd	A selector (handle) associated with "Seg#" (i.e. CODE, DATA, etc), if any.
Type	The type of Sel/Hnd (i.e.Movable Code, Fixed Data, etc), if any.

ASC [num]

Displays an ASCII table in a scrollable window. If **num** is given the row containing this value will be the first row displayed (**num** may not be in the first column). The table shows the Hex value, the Decimal value and the ASCII character these values relate to.

V86 Information

These commands provide information on various real mode (V86) resources. There are no commands for modifying these resources.

MAP

Displays a DOS memory map with the following information:

Type	The memory type if known (i.e. PSP or ENV).
Start	The V86 segment of this memory.
Len	Size of this memory in paragraphs (16-bytes).
Env	Segment of this memory's environment area.
PSP	Segment of this memory's PSP (program segment prefix).
Owner	The name of the process that this memory belongs to, if known.

To see the code associated with this memory use the **u** command. For example if **Start** is 196E, then you would use the command:

u 196E!0 (use ! because this is V86 code).

DEV

Displays DOS device drivers in a scrollable window with the following information:

HEADER	The V86 segment and offset of device driver header.
Stra	Offset from the start of the device driver to the Strategy Routine.
Intr	Offset from the start of the device driver to the Interrupt Routine.
Attr	The device driver attributes.
Device	The device driver name.

To see the code associated with this memory use the **u** command. For example if **HEADER** is 0070:008D, then you would use the command:

u 70!8D (use ! because this is V86 code).

DI [n]

Displays the DOS interrupt vectors in a scrollable window starting at 0 or **n**. Vectors in bold (bright) text have changed since SPW was loaded or since the last **DI GET** command.

INT#	Interrupt number.
WAS	Original vector address when SPW was loaded (or since a **DI GET**command).
NOW	Current vector address.
OWNER	The process that "owns" (has hooked) this interrupt, if known.

To see the code associated with this memory use the **u** command. For example if **WAS** is 0825:08DC, then you would use the command:

u 825!8DC (use ! because this is V86 code).

DI GET

Records the current value of the DOS interrupt vectors. If you follow this command with **DI** you will see that no vectors are shown in bold (bright) text and that the WAS column has been set to the same value as the NOW column for all vectors. This command allows you to mark the current vector settings so that you can detect any changes from this point on.

This command does **not** affect the vectors themselves.

Misc

These commands perform miscellaneous functions that do not fit into the other command groups.

R reg|flag [=] val

Allows you to modify a register (**reg**) or a specific eflag bit (**flag**). *Appendix C* gives the valid values for **reg**, **flag** and **val**.

R EAX 129934	Sets the value of EAX to 129934.
R EDI EDI+1	Increments EDI.
R EBX EDI+EAX+238	Sets EBX to the sum of EDI, EAX and 238.
R CF 1	Sets CF (carry flag) to 1.

H n1 [n2]

This command performs several predefined operations on the given operands. If **n2** is omitted than **n1** is used for both operands. The following information is displayed, where A and B represent **n1** and **n2** respectively:

A=	Shows the hex and decimal values of **n1**, i.e. A=00000010 A=.16).
B=	Shows the hex and decimal values of **n2**, i.e. B=00000012 A=.18).
~A, ~B	The binary not of A or B. If A=00000002 then ~A= FFFFFFFD.
-A, -B	The 2's complement of A or B. If A=00000002 then -A= FFFFFFFE.
A+B	The sum of A and B.
A/B	The quotient of A divided by B.
A&B	The binary AND of A and B.
A^B	The binary XOR (Exclusive OR) of A and B.
A-B	The sum of A and -B.
A%B	The remainder of A divided by B (binary modulus of A and B).
A|B	The binary OR of A and B.
A*B	The mulitplication of A and B (64-bit result).

LOG [ON | OFF | CLEAR | cmd]

This command controls logging of information to the Log Buffer:

LOG	Displays the current contents of the Log Buffer in a scrollable window.
LOG ON	Turns logging on. Does not destroy the current log contents.
LOG OFF	Turns logging off. Does not destroy the current log contents.
LOG CLEAR	Clears the Log Buffer.
LOG **cmd**	Logs the output of **cmd**, which can be almost any SPW command (i.e. log dev). Does not destroy the current log contents.

PRINT LF | FF | cmd

Prints a Line Feed, Form Feed or output of **cmd** to the port defined by **SET PRN**.

PRINT LF	Prints a line feed.
PRINT FF	Prints a form feed.
PRINT MAP	Prints the output of the **MAP** command.

VS

Displays the user screen. This is useful when you have only 1 monitor for both the user screen and the debugger screen. Otherwise you can already see the user screen and **VS** doesn't do anything. To return to the SPW screen, press any key.

?

Displays a list of SPW commands and a short description of each in a scrollable window.

Customization

These commands let you customize SPW to your particular needs. Most of these commands are already in the sample INI file provided with SPW. Changing these will allow your customization to be used every time you start SPW.

SET CMn[[=]adr]

Set Code Marker **n** (0..9) to the specified **adr**. If **adr** is omitted then the marker is set to the address currently at the top line of the Code Window. This command allows you to bookmark a location for later reference (using **CMn**). The address specified does not have to be valid code. The default for all Code Markers is 28:0 (selector 28, offset 0).

SET CM7	Sets Code Marker 7 to the address at the top of the Code Window.
SET CM3 = EAX+EBX:4	Sets Code Marker 3 to segment EAX+EBX, offset 4.

SET DMn[[=]adr]

Set Data Marker **n** (0..9) to the specified **adr**. If **adr** is omitted then the marker is set to the address currently at the top line of the Data Window. This command allows you to bookmark a location for later reference (using **DMn**). The default for all Data Markers is 30:0 (selector 30, offset 0).

SET DM7	Sets Data Marker 7 to the address at the top of the Data Window.
SET DM3 = GS+10:EBX	Sets Data Marker 3 to segment GS+10, offset EBX.

CMn

Goto Code Marker **n** (0..9). Change the address displayed in the Code Window to that of the specified Code Marker. If the Code Marker was not set (using **SET CMn**) the default is used (28:0, selector 28, offset 0).

CM6	Goto Code Marker 6.

DMn

Goto Data Marker **n** (0..9). Change the address displayed in the Data Window to that of the specified Data Marker. If the Data Marker was not set (using **SET DMn**) the default is used (30:0, selector 30, offset 0).

DM6	Goto Data Marker 6.

SET PRN [=] LPTx | n

Sets LPT**x** (1..3) or I/O port **n** (0..FFFF) as SPW's printer port. This is used by the **PC**, **PD** and **PRINT** commands. Make sure that the **x** or **n** you choose is a valid parallel port on your

system, otherwise doing I/O to this address may cause your system to crash.

Normally you can see how many ports are on your system and what their addresses are by using the DW command as follows: **DW 40!8**. The first 3 words displayed in the data window are your system's parallel port addresses. A value of 0000 means the port does not exist. The normal values you should expect to see are 3BC, 278 or 378. Keep in mind that it is possible for this area of memory to be modified. For example some networks duplicate a port address for remote printing. Also some applications copy this information then put all zeroes here in an attempt to control printing. Regardless of the values here, if you have a functioning parallel port and if you specify its proper address you will be able to print to that port.

SET PRN LPT1	Use LPT1 as the default printer. Its address is found at 40!8.
SET PRN LPT2	Use LPT2 as the default printer. Its address is found at 40!A.
SET PRN 3BC	Use i/o address 3BC as the default printer.

SET SCR [=] VGA [colors] | MONO [colors] | LTPx | COMx[:baud]

This command sets the type of display that SPW will use when it is active, as described below:

VGA [**nbrf**]	SPW will use the primary VGA display with the given colors (see below).
MONO [**nbrf**]	SPW will use a Monochrome display with the given colors (see below).
LPTx	SPW will use a remote display via LPT**x** (1..3).
COMx	SPW will use a remote display via COM**x** (1..4). The baud rate is given by **:baud**, in decimal, and the default is ??.

For VGA and MONO selections you can specify colors to use for 4 types of text. You do this by giving the hex value of the desired color combination, *Appendix A* lists these values:

n	Normal text. This applies to most of the information displayed by SPW.
b	Bold (bright) text. Applies to; text displayed on the frame, error messages, changed values (i.e. Registers, Flags, Interrupt Vectors).
r	Reverse video (highlighted) text. Applies to the Status Line and the instruction at the current CS:EIP.
f	Frames. Applies to the vertical and horizontal lines that partition the display.

Examples:

SET SCR MONO	Use MONO display with default colors: 07 0F 70 07 (white on black, bright white on black, black on white, white on black).
SET SCR VGA	Use VGA display with default colors: 17 1F 74 13 (white on dark blue, bright white on dark blue, red on white, light blue on dark blue).
SET SCR VGA 03 12 81 13	Use VGA display specifying your own colors.
SET SCR LPT1	Use remote display via LPT 1.
SET SCR COM1:115200	Use remote display via COM 1 with a baud rate of 115200 (decimal).

SET HOTKEY[=]scancode

Sets SPW's hotkey to scan code **n** (decimal). When this scancode (keystroke) is detected during normal system processing, SPW will be activated. A list of scancodes is too lengthy to include in this document (i.e. over 300 possibilities for an enhanced keyboard). Not all scancodes are usable ??. This command will only work from the **spwxx.ini** file. You can not change the hotkey after SPW is loaded ??.

SET HOTKEY 57	Hotkey is CTRL-SPACE (default).
SET HOTKEY 52	Hotkey is CTRL->.

SET [~][^][@]key[=]cmd[;]

Assigns the command(s) given by **cmd** to the specified key stroke. This allows you to define macros making it more convenient to perform various operations. You can use this command

in the INI file to customize SPW for your own needs. Look at the sample INI file provided with SPW for additional examples. Note that the last ';' is optional. If you do not include it you will have to press ENTER for the last command in **cmd** to be executed.

key	Can be: F1-F12, UP, DOWN, RIGHT, LEFT, HOME, END, PGUP, PGDN, FIVE, PLUS, MINUSCR, TAB, ESC, BKSP,INS, DEL, PRT, or SPACE.
cmd	One or more SPW commands.

These examples may not be useful for you, but they give you some ideas (see the INI file also):

SET ~F12 T;DW CS:EIP;	Shift-F8 will do the following: Trace 1 instruction then display the data at the current CS:EIP in the Data Window as words.
SET FIVE U EIP;MC -4;	Pressing '5' (on the number pad only) first displays the instruction at CS:EIP in the top of the Code Window. Then the Code Window is scrolled -4 instructions (lower address). This result is that the instruction at CS:EIP is displayed on the middle row of the Code Window.
SET FIVE U EIP;MC -4	Same as above, but because there is no ending ';' you must press ENTER before the **MC -4** command is executed. (The instruction at CS:EIP will appear at the top row of the Code Window. Then you press ENTER and the Code Window is scrolled back 4 instructions).

Appendix A - Color Definitions for **SCR** command

The following table shows the **hex** values of foreground and background colors. To obtain a specific color combination add a foreground and background value together. In addition you may add the Bright and/or Flashing values. Some examples are shown after the table.

Forground	(hex)	**Background**	(hex)
Black	00	Black	00
Dark Blue	01	Dark Blue	10
Green	02	Green	20
Light Blue	03	Light Blue	30
Red	04	Red	40
Purple	05	Purple	50
Brown	06	Brown	60
White	07	White	70
Bright	08	Flashing	80

White on Black = 07 + 00 = 07.
Red on Dark Blue = 04 + 10 = 14.
Flashing Bright Green on White = 02 + 08 + 70 +80 = FA.

To obtain Yellow, use Bright Brown (0E + background).

Appendix B - Hardware Breakpoints and Debug Registers

This appendix provides a bit more insight into hardware breakpoints and Debug Registers. It provides basic information for the novice and serves as a reminder for the proficient. The Debug registers are available on 80386 CPUs and above, all are 32 bits wide. These registers allow the use of hardware breakpoints and can only be accessed by an application running at priviledge level 0.

DR0..DR3 = Debug Address Registers. Each of these registers can hold one **linear** breakpoint address, allowing up to 4 hardware breakpoints.

DR6 = Debug Status Register. When the condition for one or more of the hardware breaks occurs, the CPU sets the appropiate bits of this register, indicating which breakpoints had conditions that were met. The debugger (i.e. SPW) should clear DR6 before returning to normal processing. Note that this register is updated when a breakpoint condition is met even if that breakpoint is disabled (no break occurs, but DR6 is modified).

31..16	15	14	13	12..4	3	2	1	0
0..0	**BT**	**BS**	**BD**	**0..0**	**B3**	**B2**	**B1**	**B0**

0 = Reserved, must be set to 0.
BT = Task switch occurred.
BS = Single Step Trap (Trap Flag = 1).
BD = Used when an In-Curcuit Emulator (ICE) is in use.
B3..B0 = Break caused by DR3..DR0 (if GEn or LEn = 1 in DR7).

DR7 = Debug Control Register. Sets the conditions on which the breakpoints are to occur.

31/30	29/28	27/26	25/24	23/22	21/20	19/18	17/16	15..10	9	8	7	6	5	4	3	2	1	0
LEN	**R/W**	**LEN**	**R/W**	**LEN**	**R/W**	**LEN**	**R/W**	**0..0**	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**	**G**	**L**
3	**3**	**2**	**2**	**1**	**1**	**0**	**0**		**E**	**E**	**3**	**3**	**2**	**2**	**1**	**1**	**0**	**0**

0 = Reserved, do not change.
LENn = Breakpoint memory size: 00=8-bit, 01=16-bit, 10=reserved, 11=32-bit.
R/Wn = Break on: 00=execution, 01=write, 10=reserved, 11=read or write.
GE, LE = Setting GE or LE puts the processor into "exact match" mode, which slows the system (insignificantly) to ensure the break occurs on the proper instruction instead of the following instruction. Needed when breaking on memory access instead of execution. GE is global, LE is for a single task and is cleared on a task switch.
Gn, Ln = Setting one of these bits enables breakpoint n (DRn). Again Gn is global, Ln is for a single task and is cleared on a task switch. Even if both Ln and Gn are 0 for a particular breakpoint, DR6 will still be updated if the break condition is met, although no break will occur.

Appendix C - Terminology

The following terms are used throughout this document, especially in the section *SPW Commands*.

byte = 8-bits.
word = 16-bits (2 bytes).
dword = 32-bits (4 bytes).

~ = Shift. Used when describing keystrokes, i.e. ~F8 means Shift-F8.
^ = Ctrl. Used when describing keystrokes, i.e. ^F8 means Ctrl-F8.
@ = Alt. Used when describing keystrokes, i.e. @F8 means Alt-F8.

GDTR = Global Descriptor Table Register.
GDT = Global Descriptor Table.
LDTR = Local Descriptor Table Register.
LDT = Local Descriptor Table.
TR = Task Register.
TSS = Task State Segment.
PDBR = Page Directory Base Register (also known as CR3).

reg = One the following register names:
EAX, EBX, ECX, EDX, ESI, EDI, EBP, ESP, EIP, AX, BX, CX, DX, SI, DI, BP, SP, IP, AL, AH, BL, BH, CL, CH, DL, DH, CS, DS, ES, FS, GS, SS

flag = One of the following EFLAGS (extended flags) bits:
CF, PF, AF, ZF, SF, TF, IF, DF, OF, NT, RF, VM, AC

exp = Expressions can be formed by one or more of the following:

- hex = A hexadecimal number. Leading zeroes are not required (ABCD is valid).
- .num = A decimal number. (Omitting the '**.**' treats the value as hex. Using 4509 instead of .4509 results in 17673 decimal).
- reg = A register as defined above.
- .AP = offset value of effective address.
- .AS = segment (selector) value of effective address.
- .DP = offset value of Data window.
- .DS = segment (selector) value of Data window.
- .CP = offset value of Code window.
- .CS = segment (selector) value of Code window.
- .WP = offset value of Watch window.
- .WS = segment (selector) value of Watch window.

Within an expression you may use the operators shown in the following table:

Arithmetic		**Logical**		**Boolean**	
+	add	|	or	=	equal
-	sub / neg	&	and	<>	not equal
*	mul	^	xor	>	greater
/	div	~	not	<	less than

%	mod	>=	greater or equal
[]	contents	<=	less than or equal
()	group		

sym = Symbolic address names can have following forms:

name	=	Address of a VxD service by name	(Get_Cur_VM_Handle)
		Exported name of a module	(DOS3CALL)
mod.name	=	Exported name in a module	(KERNEL.OPENFILE)
mod.nnnn	=	Exported ordinal (hex) in a module	(KERNEL.1d)
vxd.name	=	VxD service by name	(VMM.OPENFILE)
vxd.nnnn	=	VxD service by number (hex)	(VMM.E0)
vxd.PM	=	Address of the PM API of a VxD	(VTD.PM)
vxd.V86	=	Address of the V86 API of a VxD	(VDD.V86)
vxd.CP	=	Address of the Control Procedure of a VxD	(VDD.CP)
vxd	=	Same as vxd.CP	

adr = Address expressions are formed by a selector (segment) **exp** and an offset **exp** the default selector (segment) is the one currently displayed in the Data Window:

exp1:exp2	=	Uses segment (if PM) or selector (if V86) and offset.
exp1\exp2	=	Uses PM mode selector and offset address
exp1!exp2	=	Uses Real/V86 mode segment and offset address
exp	=	Uses current segment (shown in Data Window) and mode
sym	=	Symbolic name (DOS3CALL)

If the segment part is not defined, the implied segment will be used (if any).
If the segment is undefined, the current code/data will be assumed.

blk = Specifies a memory block as follows:

adr L len	=	Starting address and length, the L is literal:	PD DS:EDX L 20
adr end	=	Starting address and ending offset:	PC CS:2000 207E

str = Data strings can use any combination of the following:

exp	=	Numeric value
'text'|"text"	=	ASCII value

cond = Count and condition parameters for a breakpoint:

N=n	=	Break on every **n**th occurence, **the default is N=1**.
exp	=	Break when **exp** when expression is true, i.e. AH>3D.
N=n exp	=	Combination of the count and expression, N=5 DS=AF.

Remember that all numbers are hex unless they are prefixed with a "."!

Appendix D - FPU Registers

This appendix explains the formats and gives a **basic** explanation of various FPU registers.

Floating Point Registers
There are 8 floating point registers, referred to as ST(0)..ST(7). Each is 80-bits wide and can use 3 different formats as shown below:

Short Real Format (32-bits)

31	30..23	22..0
sign	**exponent**	**fraction**

sign = 0 is positive, 1 is negative.
exponent = Interpretted as $2^{(exponent - 127)}$. This is the binary multiplier of the fraction.
fraction = There is always an implied 1 in front of the fraction so the real value of this field is 1.fraction, i.e.1011 is really 1.1011 (binary).

Long Real Format (64-bits)

63	62..52	51..0
sign	**exponent**	**fraction**

sign = 0 is positive, 1 is negative.
exponent = Interpretted as $2^{(exponent - 1023)}$. This is the binary multiplier of the fraction.
fraction = There is always an implied 1 in front of the fraction so the real value of this field is 1.fraction, i.e. 1011 its really 1.1011 (binary).

Temp Real (Extended Precision) Format (80-bits)

79	78..64	63..0
sign	**exponent**	**fraction**

sign = 0 is positive, 1 is negative.
exponent = Interpretted as $2^{(exponent - 16383)}$. This is the binary multiplier of the fraction.
fraction = There is **not** an implied 1 in front of the fraction in this format, i.e.1011 means 1011 (binary).

In all formats an exponent consisting of all 0 bits or all 1 bits is a special case. For example, all 0 bits for both the exponent and the fraction represents the value 0 (can be +0 or -0). Other cases are beyond the scope of this document.

Status Word Register
Indicates the current status of the FPU and identifies any exceptions.

15	14	13..11	10	9	8	7	6	5	4	3	2	1	0
B	**C3**	**TOP**	**C2**	**C1**	**C0**	**ES**	**SF**	**PE**	**UE**	**OE**	**ZE**	**DE**	**IE**

- **B** = The FPU is busy executing an instruction or an exception is being indicated in bits 0..5. The instruction FNSTSW can be used to examine these bits.
- **C3..C0** = Affected by various operations, the scope is outside this document.
- **TOP** = The value here (0..7) indicates ST(n), which is the current top of stack.
- **ES** = Unmasked Exception.
- **SF** = Stack Fault.
- **PE** = Precision Exception.
- **UE** = Underflow Exception.
- **OE** = Overflow Exception.
- **ZE** = Divide by Zero Exception.
- **DE** = Denormal Exception.
- **IE** = Invalid Operation Exception.

Control Word Register
Indicates the current status of the FPU and identifies any exceptions.

15	14	13	12	11..10	9..8	7	6	5	4	3	2	1	0
X	**X**	**X**	**0***	**RC**	**PC**	**X**	**X**	**PM**	**UM**	**OM**	**ZM**	**DM**	**IM**

- **X** = Not used
- **0*** = 80287 **only**: 1 uses positive and negative infinity. 0 uses an undefined infinity. Other FPUs ignore this bit.
- **RC** = Determines how to round values that can't be represented exactly:
  00 = Round to nearest (uses an even number if exactly in the middle), default.
  01 = Round up (toward negative infinity).
  10 = Round down (toward positive infinity).
  11 = Round toward zero (truncate).
- **PC** = Determines the format:
  00 = Single precision (Short Real).
  01 = Reserved.
  10 = Double precision (Long Real).
  11 = Extended precision (Temp Real), default.
- **PM** = Precision Exception Mask.
- **UM** = Underflow Exception Mask.
- **OM** = Overflow Exception Mask.
- **ZM** = Divide by Zero Exception Mask.
- **DM** = Denormal Exception Mask.
- **IM** = Invalid Operation Exception Mask.

Tag Word Register

Each group of 2 bits relates to an internal Tag Register (T0..T8). This register is provided so status can be obtained if needed.

00 = Tag Register value is valid.
01 = Tag Register value is 0.0.
10 = Tag Register value is infinity, a denormal, or invalid.
10 = Tag Register is unused.

Appendix E - References

I just happened to have these books available when I was doing the documentation. I'm sure there are many others, but these served my purpose and it's only fair to give them credit.

386^{TM} DX Microprocessor Programmer's Reference Manual by Intel
published by Intel Corporation 1990, ISBN 1-55512-131-4
This book will give you details on CPU Architecture, Protection Mechanisms, Exceptions, etc.It will also show the formats of various Registers, Gates, Descriptors, etc. Finally it has a detailed section on the 386 Microprocessor Instruction Set, including priviledged instructions.

Microsoft's 80386/80486 Programming Guide by Ross P. Nelson
published by Microsoft Press 1991, ISBN 1-55615-343-0
This book will give you details on CPU Architecture, Protection Mechanisms, Exceptions, etc.It will also show the formats of various Registers, Gates, Descriptors, etc. Finally it has a detailed section on the 486 Microprocessor Instruction Set, including priviledged instructions.