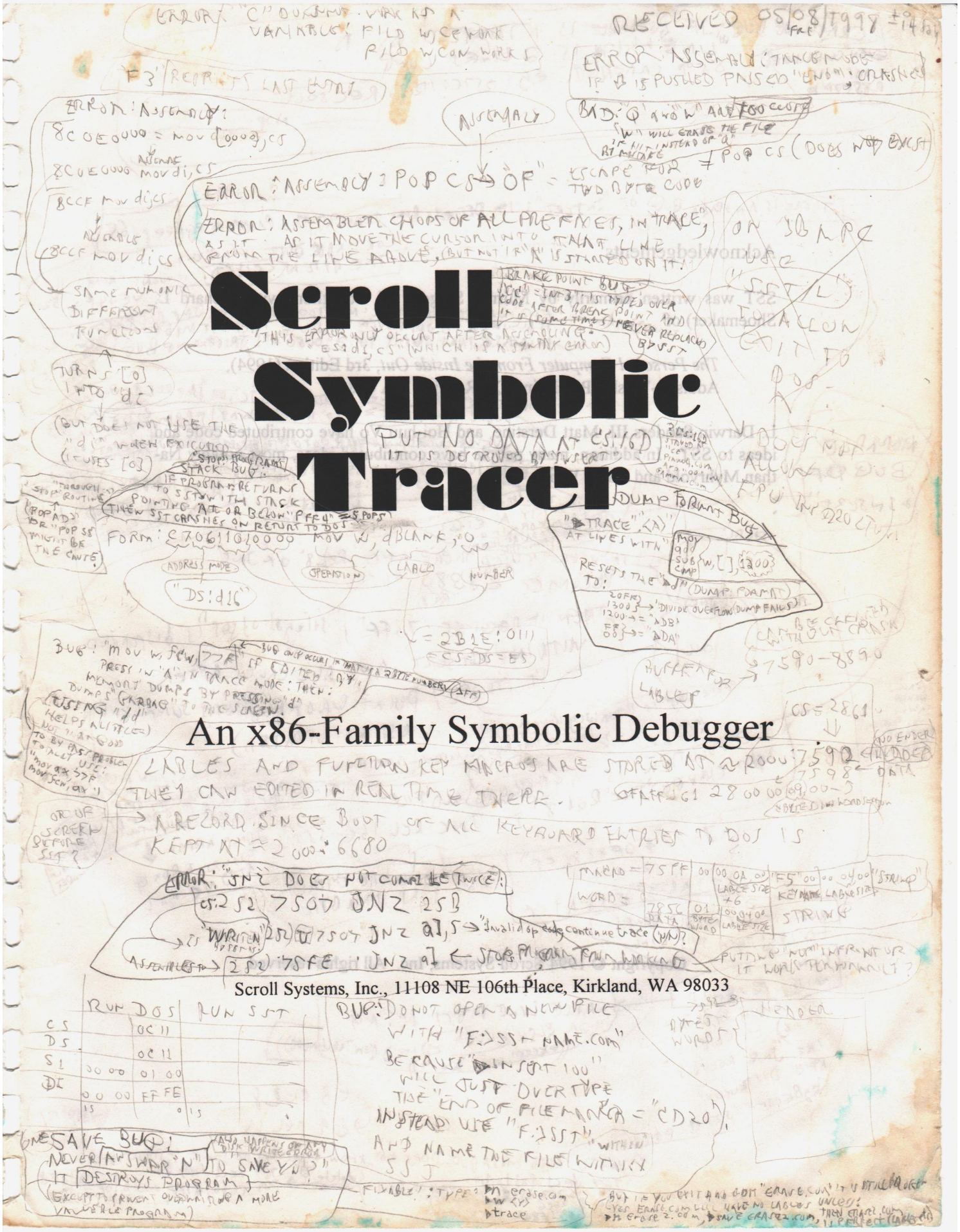
Scroll Symbolic Tracer

An x86-Family Symbolic Debugger

Scroll Symbolic Tracer



An x86-Family Symbolic Debugger

Scroll Systems, Inc., 11108 NE 106th Place, Kirkland, WA 98033

Acknowledgements

SST was written primarily by Murray Sargent III, coauthor (with Richard L. Shoemaker) of

The Personal Computer From the Inside Out, 3rd Edition (1994),
Addison-Wesley Publishing Co, Reading MA 01867

L. Darwin Sanders III, Matt Derstine, and Hoi-huu Vo have contributed code and ideas to SST. In addition, many people have contributed ideas, most notably Nathan Myhrvold and David Weise.

Table Of Contents

1. Introduction to SST

If you work with or want to learn x86 assembly language, SST is for you. Similarly if you need to debug programs written in assembly language, SST can be invaluable. SST is a screen-oriented, upward-compatible replacement for the ubiquitous DEBUG.COM distributed with MS-DOS. Use SST like DEBUG and enjoy access to a relaxed syntax, numerous extensions, ready help information (just type Function Key 1), and instantaneous full screen displays. In particular, the trace and display functions are much more powerful than DEBUG's. SST is to DEBUG much as a word processor like a screen editor is to a line editor. SST includes support for the Intel x86-family microprocessors up through the Pentium and runs on all MS-DOS systems since version 2.0 (although we haven't checked out recent versions on MS-DOS 2.0!).

1-1. SST, The Integrated Debugger

SST is an integrated debugger that combines RAM, disk, screen-font, and code display facilities with syntax is used for all modes, making them easier to use than a set of unrelated programs. SST also incorporates an assembly language interpreter that allows you to write and debug com files much as you create BASIC files using a BASIC interpreter. The com files so generated can run stand alone or under SST's supervision and run at full machine speed, unlike other interpreter code. Type a or A to run the Auto demo of the Function Key 7 to see how the interpreter and other features work and consult Chap. 6 for further information about the interpreter.

SST runs in both resident and nonresident modes. If you can afford the extra RAM, you can run a resident copy to give you instant access to the built-in calculator, system extensions, ready debugging, and trapping of errors such as divide overflow. Enter SST at any time by typing Ctrl-Enter or pressing an NMI button. Nonresident use is valuable for debugging programs and running the interpreter.

This manual is primarily a reference to SST and does not have much tutorial material. We recommend you take a guided tour through many of SST features by running the Auto demo offered when you type Function Key 7 in the SST COMMAND MODE. You might also find one or more of the many books available

on x86 assembly language helpful. In particular, the book *The Personal Computer from the Inside Out*, 3rd Ed (1994) by Sargent and Shoemaker (Addison Wesley) contains several chapters on assembly language and uses SST to illustrate various PC principles. The "Sample Program" section of Chap. 2 helps to explain how to load and trace a program using labels.

This first chapter introduces many of SST's features and explains how to use this manual. Subsequent chapters explain the features in greater detail.

1-2. Full Screen Display Mode

The display command displays a delimited area of memory if both the start and end addresses are given. However if neither or only one address is specified, an instantaneous full screenful of memory is displayed. This screen can have the usual hex/ASCII format or a pure ASCII format. The cursor arrows, PgUp, PgDn, space bar, and backspace move the cursor around. It is possible to scroll rapidly through all of memory scanning for text. A variety of hot keys allow you to use the information at the cursor as pointers to move around memory and to define blocks on which to operate. For a demonstration, run SST, type Function Key 7 followed by d or D. Type Function Key 1 for context-sensitive help on the display mode. More complete discussion is given in Chapters 3 and 5.

1-3. Trace Mode

The trace command allows streamlined screen-oriented execution of programs in single step or under control of breakpoints. Single-stroke hot keys are used to advance execution. The current instruction (at cs:ip) is highlighted by a reverse video bar. Whenever execution goes outside the instructions displayed, the screen is instantaneously redrawn with the appropriate new instructions. A conditional jump or loop that will be successful is identified by an arrow pointing in the direction of the jump and the target offset is displayed in boldface if it's on the screen. A small display window can display a selected portion of memory (arrow keys, PgUp, and PgDn can scroll this window, and Ctrl-U and Ctrl-D change its size), or it can track the memory locations referenced during the trace. A program stack window displays RAM starting at the top of the stack (given by the register pair ss:sp) and identifies the stack words by one of three readout offsets.

Function Key 5 zooms the stack and display windows into DISPLAY MODE, where you can overtype their RAM. Similarly the Edit hot key lets you overtype the register values. Function Key 6 moves the cursor from one window to another, allowing you to scroll the display, stack, program output, and trace windows. When in the trace window, the cursor is used for setting breakpoints, starting assemblies, and moving the instruction pointer. When the cursor is in the

program-output window, Ctrl-U moves the window height Up, while Ctrl-D moves it down. When the cursor is in any other window, Ctrl-U and Ctrl-D move the memory-examine window height Up and Down, respectively.

In continuous trace mode, the tracking-memory mode produces an impressive dynamic screen display that often reveals how a program works. The multiple-step Undo option is particularly valuable. This allows you in effect to execute backwards, discovering why registers or memory locations got their values, or how you got to the current instruction. With it you can single-step into a subroutine, change your mind retracing backwards, execute the subroutine at full speed, and continue single stepping afterwards. For an example of how to trace your own programs, see the "Sample Program" section of Chap. 2. Chapter 5 describes the TRACE MODE further.

1-4. Trace Mode Demonstration

For a demonstration, run SST, type Function Key 7 followed by t or T. This starts continuously tracing a built-in piece of code. You can pause execution by typing the space bar. Subsequent space bars single step execution; i.e., advance execution one instruction at a time. Other single stroke commands include those to single-step, break at the current instruction, break at the current instruction after executing it a specified number of times, fast execution (e.g., call or loop) at full machine speed, slow execution which allows single-stepping int calls (normally executed in Fast mode), and no execution useful for skipping unwanted instructions. A program can be traced continuously as in the T demo with full screen updates, or run three times faster in a Quiet mode that only updates the register values. These continuous trace modes are interrupted by typing a character or by a reference to a memory location protected by the p command. The continuous mode is also interrupted by an illegal op code or by an op code belonging to a higher-level microprocessor (e.g., an 80286 instruction executed on an 80186 or 8088). Type Function Key 1 to see a help screen defining the TRACE MODE hot keys.

Back Tracing

SST also lets you backtrace program execution up to twenty steps by default. This can be very handy when you find the program somewhere and can't figure out how it got there. Just type u or U for Undo and watch your program execute backwards in time. To change the number of backtrace steps use the /Un switch when starting SST (see Command Line Parameters in Chap. 3). Note that the backtrace (undo) feature cannot undo values output to an I/O device.

1-5. x86/x87 Support

The assemble and unassemble commands recognize all x86-family mnemonics up through the Pentium. The search command can search for assembly language instructions as well as hex bytes and string literals.

SST fully supports the x87 numeric coprocessor with stack displays in TRACE MODE. Registers and memory can be changed by simple assignment statements using ordinary scientific notation, and all status information is displayed.

1-6. Labels

SST supports the full link MAP for the DOS LINK.EXE linker. By reading in the map (specify /MAP option on the LINK.EXE list file entry), you can refer to program line numbers and external labels (declared extrn in .asm files and external names in general declared in compiler source files). See the load label option in Chap. 5. You can also read in variable names for one segment. Variable names are not treated as generally as desired, partly due to the need for telling SST what segment should be assumed for variable references. Something like the masm.exe assume directive is needed. SST also reads in Microsoft CodeView .exe externals generated with the /co link switch.

1-7. Calculator

A Polish suffix hex calculator is included. String literals and decimal values (indicated by a decimal point) are supported in the HEX CALCULATOR, SEARCH, and ASSEMBLER MODEs. Register variables can be used in calculator expressions, and register and flags can be assigned values by direct assignment.

Command editing supports the Function Key 3 DOS Edit function (repeat to end of last command), although DOS is bypassed for all operations other than disk. This allows most DOS functions to be traced and leads to much faster response. In addition, the left and right arrow keys, Home, End, Del, and several Ctrl keys can be used for editing (see Chap. 4).

1-8. Synergy

The program serves both to teach people new to assembly language how the machine works, and to aid the advanced programmer in finding program bugs. It typically requires 1/20th the time to find a bug with SST as compared to DEBUG or SYMDEB, and sometimes a few minutes with SST can literally save you days of debugging with DEBUG. The program runs in about 100K RAM and is written

in optimized 8086 assembly language (some 80286/80386 code is used in special sections off limits to smaller microprocessors). It has many features not found in debuggers requiring two or more times as much memory. SST's relatively small size per feature is due partly to tight coding, and partly to a careful integration of facilities that allows the various components to take advantage of one another (synergy!).

1-9. SST Windows

SST displays a number of tiled windows that depend on the mode of operation. Initially SST operates in COMMAND MODE. Depending on the command chosen, SST may switch into one of several other modes, namely ASSEMBLE MODE, ASSEMBLE INSERT/EDIT MODE, UNASSEMBLE MODE, TRACE MODE, XAMINE MODE, DISPLAY MODE, OVERTYPE MODE, DISK DISPLAY MODE, DISK OVERTYPE MODE. The Enter key (↵) returns to COMMAND MODE (↵↵ return in ASSEMBLE MODE).

Menu Line

The current mode is displayed on the menu line at the top of the screen. For example, in COMMAND MODE, the menu line reads

```
COMMAND MODE: F1 0-9 Asm Cmp Dsp Exam Fill Go In Klr Ld Nam O-Z
```

The menu line is also used to report some errors and special conditions. The menu line can be toggled on and off with the r2 command.

Register Window

A register window is usually displayed at the top of the screen just below the menu line. In COMMAND MODE, the command r0 toggles this window on and off, and in TRACE MODE, the hot key 0 turns it on and off. Other windows appear for various commands, and pop-up help screens always appear immediately below the register window. This special window has the 8086 form:

```
ax=0000 bx=0000  ds=4F97 es=4F97 cs=4F97 ss=4F97  bp=0000
dx=0000 cx=0000  si=0000 di=0000 ip=0100 sp=FFFE  NS Z NC
```

The numbers in this figure represent 4-digit hexadecimal numbers. The third line in general displays the menu for whatever mode is currently active. Here the COMMAND MODE menu is showed in part. This particular menu is displayed

when SST is started and whenever you type a ↵, which stands for the Enter key (type ↵↵ in ASSEMBLE MODE).

The register window groups the registers according to their typical usage in 8086 code. The ax, bx, cx, and dx registers are the general accumulators that can also be split into pairs of 8-bit registers like ah and al. The segment registers ds, es, cs, and ss are shown directly above the 16-bit registers with which they are commonly paired. Specifically, the addresses ds:[si] and es:[di] are used with the powerful 8086 string instructions. The cs:[ip] address gives the current instruction, and the ss:[sp] address gives the top of the program stack. In addition, ds:[bx] and ss:[bp] are common addresses, so it is handy to have the corresponding registers near one another.

On the top line the ds:[n_1]=n_2, which appears only when a memory reference occurs and displays the value and address of such a reference. If the value is a byte value, only two hexadecimal digits are displayed.

After the sp=*n* field on the second line, the flag values are displayed. For example, if the Zero flag is set to 1, you see "Z". If it is reset to 0, you see "NZ" as shown in the figure. This notation corresponds to the instruction mnemonics used by the unassemble and trace commands. Note that since the TRACE MODE reverse video bar for the current instruction indicates whether a conditional jump will occur, it isn't nearly as important to consult the flags as it is with other debuggers. PE means that the last instruction that affects the parity flag found Even Parity, while PO stands for Odd Parity. A + or - indicates the direction in memory that repeated string operations go. The instruction cld (CLear Direction) gives a plus sign (+), which is the usual direction for most programs. If Interrupts are Enabled, you then see EI, while if they are Disabled, you see DI. The two remaining flags, OV (Overflow Flag) and AC (Alternate Carry) occur less often and are only displayed if they are on. This choice helps to reduce screen clutter and separates the principle set of flags (Sign, Carry, and Zero) from the others.

Following the Interrupt flag value, four SST status values are displayed in reverse video if their corresponding functions are enabled. Indicating reverse video by an underline, the status values are E for active Echo output (see n> command), S for active Super-Trace conditions (see trace command), T for Tracking memory display window (see trace command), and V for x86 protected Virtual address mode (see vm command).

80386 Register Window

By default on 80386-based computers, the register window displays the complete 32-bit 80386 register values in the form

```
eax=0000 0000  ebx=0000 0000  ds=4F97  ss=4F97  ebp=0000 0000
edx=0000 0000  ecx=0000 0000  fs=4F97  gs=4F97  esp=0000 FFFE
esi=0000 0000  edi=0000 0000  es=4F97  cs=4F97  eip=0000 0100  NS Z NC
```

In COMMAND MODE, the r3 command and r1 switch to the 80386 and 8086 register sets, respectively, while in TRACE MODE, the hot keys 1 and 3 perform these switches. The dr command displays special 80386 registers.

1-10. Disk Editor

SST contains a disk editor invoked by the command disk in COMMAND MODE. The idea is that in place of the segment specification for RAM, you type a sector number. The facility, described further in Chap. 7, has a variety of options to facilitate moving around a disk.

1-11. Mouse Support

The DISPLAY and TRACE MODEs can use the mouse to move the cursor around. To enable the mouse, you have to run the appropriate MOUSE program at the DOS command level, and then tell SST that it should use the mouse by typing the mouse command in SST's COMMAND MODE. The mouse allows you to move around the display screens rapidly and to edit character fonts.

1-12. Super-Trace

SST has a pair of powerful conditional break facilities for advanced users. The first is the Super-Trace mode, which traces program execution at about one-tenth full speed (MS-DOS running in real mode, not V86 mode) and after each instruction it checks an arbitrary set of conditions specified by the user in assembly language. If the result of these conditions sets the Zero flag, tracing is halted; otherwise the trace continues. This allows a very rapid execution search for any desired machine state. It implements in software features that can usually be performed only by expensive hardware tracing boards, and has generality that the hardware methods cannot match. The user conditions can even call user-supplied subroutines, allowing specialized monitoring such as program execution profiling. The use of ordinary assembly language for the user conditions combines the highest execution speed, the simplest implementation and documentation, and the greatest power available in the computer. Because of the great flexibility of the method, you have to be careful not to include a command that will crash the computer. Hence we consider the Super-Trace to be a facility for advanced users, although simple Super-Traces can be run by beginners (see Chap. 3 demonstrations).

1-13. Conditional Breakpoints

Alternatively, breakpoints can be associated with the same arbitrary set of conditions as the Super-Trace. For these, execution proceeds at full speed until the computer attempts to execute the instruction at one of the user-defined breakpoint locations. The user's conditions are then checked. If they succeed in setting the Zero flag, program execution is halted and control is returned to SST. Otherwise execution proceeds again at full speed. If no breakpoint is encountered, you can usually recover control by typing Ctrl-Enter.

1-14. How to Use this Manual

This manual tells you how to use SST. It should be used in combination with a book or reference manual on assembly language for the Intel x86 microprocessors. The book *The Personal Computer from the Inside Out* by Murray Sargent III (SST author) and Richard L. Shoemaker (Addison-Wesley Publishing Co., 3rd Edition, 1994) is one of several such books. If you are already familiar with assembly language and debug.com, you may just want to glance at this introductory section, at Chap. 4 on Syntax, and then refer to the command descriptions of Chap. 5 when the built-in help messages are too terse. If you are learning assembly language, read the Help section (Chap. 2), the Demonstration section (Chap. 3), run the Auto demo of Function Key 7, read the Syntax section (Chap. 4), and read your book on assembly language. Try out the built-in demonstrations to get a feel for how memory looks and how a program runs. Assemble some simple code of your own and trace its execution with the trace command. You'll learn assembly language in a fraction the time required by traditional methods.

References

Many excellent assembly-language and higher-level language books are available in your favorite bookstore. For example, Dalton's carries a large number of relevant books or check the references in Sargent and Shoemaker (1994). Browse around a bit and choose the books that seem to be the most helpful.

2. SST Help Facility

In all modes, typing Function Key 1 displays an appropriate help screen in a pop-up window just below the register window. Typing any key (except for Function Key 1 itself in ASSEMBLE MODE) replaces the screen text that was covered up by the help window. In particular, typing Function Key 1 instead of a command displays the menu

!&.<>	0-9	@scii	Asm	Baud	Comp	Display
Exam	Fill	Go	Hex	In	Klear	Load
Move	Name	Out	Protect	Quit	Reg	Search
Trace	Unasm	Vector	Write	Xam	YGDT	Zam

For a command description, type command letter followed by F1

exp = {*value* | exp_1 exp_2 *op*}
address = [*segment*:] *offset* *range* = $address_1$ $address_2$

This gives the names of the simple commands available under SST. To run a command type the first letter of the command name followed by appropriate arguments. If you type Function Key 1 in the middle of typing a command, a terse help message for that command is displayed in a pop-up window below the register window. Typing any character gets rid of the help window, replacing the text it covered up. If you type the command character immediately followed by the Function Key 1, then the command line is erased when the help window goes away. If you type more characters on the line, the command line remains, ready for further typing. This allows you to get help whenever you need it. For more information on each command, please see the corresponding page in Chap. 5, which is ordered alphabetically by command.

The built-in Function Key 1 help summaries are (for the complete a F1 displays, see assemble command below):

2-1. Brief Command Definitions

and *range list*	And (bitwise) memory in *range* with *list*
a [*address*]	Assemble. Op codes (8086 in lower case, F1 → x87) are: (see a command in Chap. 5)
a [*address*]	Assemble. x87 op codes (F1 → x86) are: (see Chap. 5)
b *rate* [,*channel*]	Set the baud rate of serial channel
bc {*list* | *}	*c* breakpoints in *list*, where *c* = c, d, e, for Clear, Disable, Enable, respectively
bl	List breakpoints
bs[*n*] *address* [*m*]	Set breakpoint [*n*] at *address* [skip *m* passes] Note: b breakpoints are sticky unlike g's
cd *path*	Change Directory to *path*
cls	Clear Screen
close	Close all files
c *range address*	Compare memory in *range* to memory at *address*
cpu	Display info about computer
date	Display date
dir [*filespec*]	Display filenames matching *filespec*
d [*address*]	Display full screen of memory
d *address*$_1$ *address*$_2$	Display memory from *address*$_1$ to *address*$_2$
d/c	Can echo to file - see n>
d/cs	display labels (*c*=segment paragraph), variable names (*c*=v), or user strings (*c*=u)
dos *n*	Execute DOS (int-21h) function ah=*n*
echo ON"	Toggle screen echo on/off
erase *filename*	Erase file *filename*
e *address*	Examine *address* in Byte mode
e *address* [/*type*]	Examine *address* by *type* = b, d, i, l, o, q, s, t, for packed BCD, Double float, Integer, Long int, Quad int, O binary, Single float, Temp float, respectively (Needs x87)
e *address* [/*string*]	Examine with structure template *string*
f *range list*	Fill memory in *range* with *list*
g [=*address*] [*address*$_1$]...	Go execute at cs:ip or =*address* with breakpoints at *address*$_1$, *address*$_2$, ... Same followed by @ allows conditions to be typed

h *value*	Convert **h**ex *value* to binary
h *value*$_1$ *value*$_2$	*Calculate value*$_1$+*value*$_2$ and *value*$_1$ - *value*$_2$
ini	Run first **s**st.ini file in DOS path
i *portaddress*	**I**nput byte from *portaddress*
int21 [*n*]	Display int-21 function definition(s)
j	(No command)
k	Program stack trace
k *n*	Klear next *n* lines
k *n m*	Klear from line *n* to line *m*
kf	Klear floating point (x87) registers
ki ↵ *c*	Display keyboard input code *c*
list [*address*]	Unassemble screenful starting at *address*
llist [*address*]	Unassemble to printer starting at *address*
l [*address*]	Load file named by n at **cs**:100 or *address*
l *address drive sector count*	**L**oad *count* sectors from absolute *sector* at address *address*
ll	**L**oad program **l**abels (use n to **n**ame .map file)
lm	**L**oad program labels for a .**com** file
lv	**L**oad **v**ariable names (name .lst file)
m *range address*	**M**ove memory in *range* to *address*
n *filespec*	**N**ame load or write file by *filespec*
n> *filespec*	**N**ame echo file by *filespec*
n>	Toggle echo to file
n=	Display current **n**ame file
n *string* = "..."	Define user *string* (2 letter names)
new	Reset labels and paramters to starting values
not *range*	**N** (bitwise) memory in *range*
or *range list*	**O**r memory in *range* with *list*
o *portaddress list*	**O**utput *list* to *portaddress*
pause *n*	Pause a time proportional to *n*
prompt	Toggle path prompt
p	Turn off memory **p**rotection
p *address*	**P**rotect memory *address*, i.e., Stop trace if memory *address* is referenced
p *range*	Stop trace if memory *range* is referenced
q	Quit - return to DOS

q *c n*	Set screen attribute for window *c* = a, h, n, r, s, x, z for Assemble, Help, Normal, Register, Stack, Xam, Zam, respectively
qg *n*	Set SST video RAM segment = *n*
qi *n*	Set SST interrupt mask = *n*
ql *n*	Set lines/page = *n*
qo *n*	Set SST display origin to line *n*
qol	Display in lower half of 66 line screen
qp *n*	Set 6845 CRT I/O port = *n*
qs	Swap IBM displays for SST alone
qs1	Swap displays for both SST and DOS
qs3 (2)	Turn screen save on (off)
qu *n*	Set undercover debugger port = *n*
qy *n*	Set # lines Xam window = *n*
r [*register*]	Display [change] registers. Can change registers by =
rr	Restore registers to initial values
rm	Go to real *address* mode
rn	Return NMI interrupt to preceeding owner
ren *file*$_1$ *file*$_2$	Rename *file*$_1$ *file*$_2$
s *range list*	Search memory in *range* for *list*
s *range* @	Search *range* for assembly language
s *range*	Search *range* for last string
s	Repeat last search
system	Quit to DOS
time	Display time
t [*address*]	Trace program starting at cs:ip or *address*
t @	Specify Super-Trace conditions
type *filename*	Type (browse) file *filename*
u [*address*]	Unassemble code at last address or at *address*
u *range*	Unassemble code in *range*. Use n> to echo to file
v *n* [ax [bx [cx [dx]]]]	Execute interrupt vector *n* giving optional register values
vm	Switch to protected virtual address mode (x86 x > 1 only)
width *n*	Set screen width (40 or 80)
w [*address*]	Write file named by n from cs:100 or *address*
w *address drive sector count*	Write absolute sectors *sector*$_1$ *thru sector*$_2$ *from address*
wl	Write labels

x *address*	Start trace examine window at *address*
xor *range list*	Xor memory in *range* with *list*
y	List GDT entries, one/space bar
y *n*	List GDT entry *n*
y *n address* [*access* [*length*]]	Define GDT entry 68<*n*<D8 at *address*, access = *access*, length = *length* (only works when system is in real mode)
z	ezamine x87 status

For definitions of command syntax and words like address, see Chap. 4 on Syntax. Typing \ followed by Function Key 1 displays the current disk drive:directory. To obtain this information continuously in COMMAND MODE, type the prompt command.

Typing a decimal digit followed by Function Key 1 displays help for the calculator (see Chap. 5 for more information):

Hex number	Convert to decimal
decimal number	Convert to hex
exp₁ exp₂ op	Calculate *exp₁ op exp₂* (*op* = +-*/&!)

2-2. ASCII Chart

When debugging it is often very handy to have ready access to the ASCII codes. These are usually instantly available in a pop up screen by typing @. In some situations the @ would be used for other purposes, such as in an assembly language comment, searching for assembly language, and supertracing. Hence @ doesn't give the pop up menu as the second or later character of the command line. It also works in most non COMMAND MODEs, and shows you a hexadecimal display of all 256 extended ASCII codes. Type any key other than another @ and you're back to the screen displayed before you typed the @.

The @ option has four pop-up screens. To get to the next one, type @. The pop-up screen following the initial hexadecimal ASCII screen is a decimal ASCII display. The third screen is an EBCDIC (Extended Binary Coded Decimal Interchange Code, an old code that used to be used on IBM mainframes) display with hexadecimal codes, and the fourth screen is an EBCDIC with decimal codes. Further @'s repeat this sequence of four screens.

2-3. Sample Program

To illustrate the loading and tracing of a program, the SST distribution diskette includes a simple program to get and display console input. The program is as follows:

```
public    ci,co,console_loop

;Simple console echo program that illustrates
;SST label facility

CR        = 13
LF        = 10
cseg      segment
          assume cs:cseg

console_loop:
          call    ci          ;Get next character
          mov     dl,al       ; from console
          call    co          ;Display character
          cmp     dl,CR
          jnz     console_loop
          mov     dl,LF       ;If CR, output
          call    co          ; LF automatically
          jmp     console_loop

ci:       mov     ah,7        ;21h direct console
          int     21h         ; input without echo
          ret

co:       mov     ah,2        ;21h display output
          int     21h
          ret

cseg      ends
          end     console_loop
```

You can run the sample CONSOLE program either by typing it in in ASSEMBLE MODE (see assemble command in Chap. 5), or by assembling and linking it with an assembler. For the latter you need to have an assembler available in your current directory or in some subdirectory specified by the path command in

your AUTOEXEC.BAT file. If you don't know about the path command, go read about it in the DOS manual, since it can simplify your life considerably.

Suppose for the sake of illustration that the DOS prompt is C:\> and SST's prompt is ▷. Then at the DOS prompt type

```
C:\>masm console;
C:\>link console,,console/map;
C:\>sst
▷nconsole.map
▷ll
▷nconsole.exe
▷l
▷t
```

This puts you into the SST TRACE MODE all ready to trace your simple console program. SST allows you to see what you program displays on the screen in several ways. For the present case, just type the TRACE MODE **W** option, to give yourself a DOS window on screen. Then start single stepping your way through the program by typing the space bar. When you reach the **int 21h** for the ci subprogram, the console pauses to let you type in a character. Type something other than the space bar, so that the co routine will display something you can see in the DOS window. Notice that the ASCII code of the character you type for the ci subroutine is returned in the **al** register (low byte of the **ax** register). The program then moves this character into the **dl** register. You can watch this action by looking at the register window at the top of the SST display screen.

After single stepping for awhile try some of the SST options like **D** for **D**on't single-step subroutine, **B** for **B**reak when back at the current instruction, and **G** for break (**G**o) at the address you type in. Working with this simple program can teach you a great deal about the TRACE MODE. Return to COMMAND MODE at any time by typing the Enter key or the Esc key.

For simple programs like this one, the DOS window is fine, but for more typical programs, the whole screen is needed. If you have two screens on your computer, you can put SST on the one your program isn't using (see the Section on "Multiple Screens" in Chap. 3). Alternatively you can use the screen save option discussed in Sec. "Screen Save Option" in Chap. 3. In single stepping most instructions, you'll notice no difference with the screen save option enabled, but whenever you do something that SST cannot know whether the screen will be accessed (e.g., you use an explicit or implied breakpoint), you'll notice a momentary

flashing of the screen. This is because SST restores the entire screen for the user program.

Any time you want to switch to the user screen, type v or V for View program screen in TRACE MODE. To return to SST's screen, type any key.

2-4. INT21 Command

SST automatically comments some unassembled instructions, such as DOS calls (**int 21h**), x87 emulation interrupts (**int 34h - int 3dh**), and immediate byte constant instructions like mov al,41H. In addition the **int 21h** definitions are displayed when you type the **int21** [*n*] command in COMMAND MODE. If the optional *n* is present, the definition for that entry point alone is displayed. If *n* is missing the next hexadecade of **int 21** entries is displayed. These features are very handy for working with code that makes DOS calls.

2-5. Help Command

For more information, type help in COMMAND MODE. This displays the file called SST.HLP, which has a variety of help information.

3. Running SST

The first thing to do with your SST is to see it in action! Run SST and you'll see the sign on help message in the main part of the screen and the COMMAND MODE window at the top. The x86 registers are displayed in this window followed by the COMMAND MODE menu. Type Function Key 7, type t or T for the Trace demo, and stare at the continuous TRACE MODE in amazement! What you'll see is a dynamic screen trace of the execution of a program, revealing how the registers, stack, and memory referenced by the program change. In this continuous trace mode, the program executes about 40,000 times more slowly than normal, which gives you a chance to see what's going on. For comparison with the Super-Trace described below, notice how the **di** register increments slowly (due to the **stosb** instruction) as the program runs.

3-1. Demonstrations

Function Key 7 gives you access to three demonstration. The first is the trace demo we just fired up, and the other two are a display demo and an "auto demo", which is like an SST tutorial. Let's continue with the trace demo. Typically the program runs much too fast to understand what's going on, so to stop execution, type the space bar. Successive depressions of the space bar single-step the program, always showing you the latest state of the machine. You can see what effect the instructions have on the register, flag, and memory contents. The demo uses the "tracking display" window in ASCII mode, so that you always see an 80 hex byte memory window around the last memory location referenced by the program. The size of this window is programmable - see the **qy** *n* command in Chap. 5. At the right end of the second display line from the top, you'll see a T. This indicates that the memory window is in Tracking mode.

The TRACE MODE menu indicates many other options. Type Function Key 1 to see a help screen that gives brief definitions of most of these options. This help screen is also shown in Chap. 5 under the **t**race command, along with more detailed descriptions of the options. To get rid of the help screen, type any key.

Backtrace Demonstration

After you've traced program execution for awhile, type u or U for **U**ndo. This causes the program to undo its steps, literally executing backwards in time. This feature is handy when you the program ends up somewhere and you don't remember how it got there. SST cannot trace backward forever, or it would unboot your machine! Actually SST doesn't execute backwards, it just restores the preceding machine state for up to 20 backstates by default. To change this number, use the SST/U *n* option described under Command Line Parameters in this chapter.

Display Demonstration

After commands like **a**ssemble and **l**oad are executed, the Function Key 7 demo option is suppressed to prevent SST from overwriting a program you have loaded in or typed in with the **a**ssemble command. If Function Key 7 doesn't work, type new. Type Function Key 7 followed by d or D to go into the DISPLAY MODE. You can also do this at any time in COMMAND MODE by typing d or D followed by a ↵. This gives you a full screen display of memory with the register values and a menu at the top of the screen. Type Function Key 1 to see a help screen that gives a brief definition of the menu options. Chapter 5 under the **d**isplay command also shows this help screen along with more detailed discussion of the options. Type any key to get rid of the help screen.

Overtype Mode

SST's display facility has a number of other options, including Ctrl-O, which toggles between OVERTYPE and DISPLAY MODE. This mode allows you to overtype the memory location at the cursor position. If you do this by mistake, type Ctrl-U to Undo the overtype. Hopefully you didn't overtype something important, like a keyboard interrupt vector (crash!). SST allows you to do absolutely anything with your computer, so be careful. SST isn't PASCAL, which usually prevents you from doing something you might later regret. This kind of freedom is desirable since it allows you not only to identify a program bug, but also to try out a possible fix without reassembling or recompiling and relinking. This can save you considerable time, but it does require a bit of care. Also you must remember to fix the bug in your source code, or you'll experience bug déjà vu!

Try out the various options, scroll through all of memory, and learn about your machine as only hands-on interaction allows.

3-2. Command Line Parameters

When invoking SST from DOS, you typically type a command of the form

C:\>sst [/*c*] *filename other_parameters*

SST then loads the file *filename*, and places the *other_parameters* in the command line area reserved in the program prefix, just as DOS's COMMAND.COM does.

In addition you can specify several useful options as switches following the SST. These options allow specifying the total amount of SST RAM work area, the number of back states for the TRACE MODE Undo option, and making SST resident. The Resident option is described in the next section.

C:\>sst/*n*

saves 1024**n* bytes of RAM for SST. A minimum of 9K is required. No specification results in 9 K bytes (/9). More RAM is automatically allocated as needed when labels are read in.

C:\>sst/u *n*

saves room for *n* backsteps in TRACE MODE. The default for SST is 20.

Resident Operation

Sometimes it's handy to have SST in memory for ready access in the event of a problem, or just to see the ASCII chart or use the calculators. One way to do this is to follow SST on the DOS command line by the /R switch

C:\>sst/r

This loads SST and returns to the DOS prompt with SST resident. To have SST take control, type Ctrl-↵ or press an NMI button. This method doesn't let you have the chance to set up special SST features such as loading in program labels. If these other features are needed, try the q/R option described in Chap. 5. To return to your interrupted program, type Alt-X or the **go** command.

Screen Save Option

When going between SST and a program, it's handy to be able to see the screen display generated by the program. SST automatically does this. Then whenever you return to a program using a **go** command, or use the View option in TRACE MODE, you can view the program screen rather than SST's. When trac-

ing with this mode, the screen only flickers when you "fast" trace over an **int** or **call** instruction. This is because during normal tracing, SST knows what RAM is being referenced and only needs to restore that RAM for program purposes. To turn off the automatic screen-save option, use the qs0 command described in Chap. 5.

Multiple Screens

For extensive debugging it's very useful to have more than one screen. This is particularly true when debugging graphics programs. SST can be put on whatever screen you desire using various q commands. In particular, the monochrome and EGA/VGA adapters are so widespread that SST has been setup to switch very easily between the two. Type the command (▷ stands for the SST COMMAND MODE prompt)

```
▷qs
```

to switch between them without telling DOS. Type qs1 instead to switch screens telling DOS as well. Hence you can load SST from DOS on either screen and switch back and forth as the need be and exit SST on either screen.

3-3. Configuring SST

With the abundance of different screen sizes, character attributes, and other machine characteristics, it is time consuming to configure SST appropriately each time you run it. On startup, SST automatically runs the SST.INI file in the current directory. You can customize this file for the project at hand. Alternatively, SST can be reconfigured using the q commands of Chap. 5 along with appropriate DOS commands. We illustrate this procedure using the screen attribute specification which allows setting the screen attribute (color, reverse video) for various SST windows. This command is defined by

```
▷q c n
```

which choses a screen attribute *n* for the window *c* = a, h, n, r, s, x, z for Assemble, Help, Normal, Register, Stack, Xam, Zam, respectively. In particular, experimenting with the qr *n* (set register window attribute) is a great way to learn about screen attributes.

You can configure SST.EXE to your tastes by SSTing a copy of SST.EXE and typing the q commands followed by /*n*. When done, type w or W to Write out the modified SST.EXE file. For example, on a VGA display, to set up

Red background, yellow foreground register window
Blue background Xam window
Green background stack window
Red foreground assembly language
Yellow foreground normal window
Magenta help screen

type the following DOS and SST commands:

```
C:\>copy sst.exe newsst.exe
C:\>sst newsst.exe
▷qr40/n
▷qx10/n
▷qs20/n
▷qa4/n
▷qn6/n
▷qh5/n
▷w
▷q
```

Here the commands following the SST prompt ▷ are typed in SST's COMMAND MODE. The q's store the configuration information in your program NEWSST.EXE, the w or W writes the modified NEWSST.EXE to disk, and the final q command quits SST. Then type

```
C:\>newsst
```

Your file NEWSST.EXE now has these characteristics as you'll see by typing Function Key 7 followed by t or T to see the Trace demo.

SST initialization

In addition to the customization facilities described above, the COMMAND MODE **ini** command executes the first SST.INI file that it finds by doing a DOS path search.

Echoing Output to a File

You may want to save disassembled code or hex/ASCII formatted binary displays on disk files or print them. For this purpose, some SST commands can

echo what they send to the screen to a disk file of your choice. For explicit discussion, see the n command in Chap. 5.

Terminating User Programs

Program terminations of all kinds (**int 20h**, **int 27h**, and **int 21h** with **ah**=0, 31h, and 4Ch) are intercepted by SST, which then gives a menu offering to 1) Restore registers to their initial values, 2) go into TRACE MODE, or 3) return to COMMAND MODE leaving the registers as they are. The **r**estore option acts like DEBUG.COM, allowing you to rerun programs easily. The other two allow you to investigate the circumstances that led to program termination. You can also **r**estore the registers at any time by typing rr in COMMAND MODE.

On terminating execution of a program, SST closes file handles 5 through 20, and releases all memory owned by the program's Program Segment Prefix.

Super-Trace Demonstration

With SST you can Super-Trace execution at one tenth full speed (in real mode, not V86 mode) looking for a condition of your choice to occur. For example, type ↵ to return to COMMAND MODE, type t@ ↵, which transfers control to the assembler for specifying Super-Trace break conditions, and type the assembly language instruction

```
cmp di,600
```

followed by two ↵'s, which turns on the Super-Trace mode. At the right end of the second display line, you'll see a S, to indicate that Super-Trace conditions are in effect (they apply to breakpoints as well). Now things are very different. Each time you type the space bar, the trace stops only if the condition **di**=600 is satisfied! Hence the trace sometimes hesitates between single-stepping, since the computer has to execute many instructions in between. You can run the Super-Trace continously by typing c or C. To turn Super-Trace off, type ↵ to return to COMMAND MODE, type t@ followed by two ↵'s, which turns the conditions off, and then type t or T↵ to return to TRACE MODE. Notice that the S at the end of the second screen line goes away. Type c or C to start up the **C**ontinuous trace mode again, and notice how slowly the **di** register changes. This should convince you that the Super-Trace mode is really super! The Super-Trace executes about 10 times more slowly than normal, compared to the continuous trace, which executes 40,000 or more times more slowly than normal. DEBUG traces 270,000 times more slowly than normal and cannot be read when running continuously.

Another interesting supertrace is to have SST load your favorite word processor (see **l**oad command in Chap. 5) and supertrace for a condition that will never

be met like or sp,sp. Since the stack pointer is never 0 for working programs, SST will simply run your program in slow motion. To stop, type Ctrl-↵, or tell your program to quit. Super-Trace is described further in Chap. 5 under the trace command.

3-4. Calculator

In addition to the alphabetic commands described in Chaps. 5 through 7, if a decimal digit or x86 register name starts the command, the line is assumed to be calculator input to a hexadecimal Polish suffix calculator. This calculator supports 32-bit arithmetic with the arithmetic operators "+-/*", the binary logical operators "!&", the logical not "~", and the store operator "=". See also the more limited hex command, which is included to be compatible with DEBUG.COM. As an example, typing

▷9 9*↵

displays

▷9 9*= 51

Remember that we're working with hexadecimal here, so 51 is correct! Operands must be separated by at least one blank, and the operators must follow their respective operands without intervening blanks (the operators are treated as special operand delimeters). If too few operators are given, enough +'s are added. Hence to add up a list of numbers, just type the list followed by a ↵.

Converting between Hexadecimal and Decimal

Typing a single value followed by a ↵ with no operators displays the corresponding decimal value. Similarly, typing a decimal value (identified by trailing period) followed by a ↵ displays the corresponding hexadecimal value. Examples are

▷0ACE= 2766.
▷127.= 007F

To convert from hexadecimal to binary, in COMMAND MODE type h or H followed by the hexadecimal value.

Use of Register Values

To display the value of the **ds** register multiplied by 4, type

```
▷ds 4*↵
```

More complicated expressions like

```
▷0FE,4 ds*&
```

can be used. This one calculates the infix expression (4*ds)&0FEh. If **ds**=0BCh, this gives 0f0h. Note that a leading 0 is required to indicate that 0FEh is a number and not a fill command.

To store the value of an expression into one of the x86 registers, use the = operator at the end of the expression. For example, to set **bx** = 2***ax**+**cx**, type

```
▷ax 2* cx+ bx=↵
```

For simple changes to the registers, you can just assign them values directly, as in

```
▷ax = 3
```

3-5. Interrupts

Hardware and software interrupts play important roles in x86-family computers like the IBM PCs. Hardware interrupts are used to maintain the date and time of day, keyboard buffering, some disk control operations, x87 numeric coprocessor exceptions, and optional serial and parallel input/outut data transfers. Software interrupts are used to connect programmer routines with operating system routines, to handle arithmetic exceptions such as divide overflow, and to handle single-step and breakpoint operations. To control these operations effectively, SST takes over many of these interrupts and restores them to their previous values upon returning to DOS. For example under IBM DOS an alleged DIVIDE OVERFLOW interrupt is identified as such and the machine either halts or returns to DOS, in either case preventing you from finding out where and whether an overflow occurred or whether a software interrupt occurred instead. SST gives the same message if a divide overflow really did occur and in any event leaves you pointing to the instruction that caused the interrupt. You can then investigate the conditions that caused the problem and return to DOS to correct your program accordingly.

More specifically, unless you specify **/L** (for tread Lightly) when invoking SST on the MS-DOS command line, SST takes over interrupts 0 (divide over-

flow), 1 (single-step), 2 (nonmaskable interrupt), 3 (breakpoint), and 4 (overflow). On PCs with 80287 and later processors, SST also takes over 5 (for trapping the **bound** instruction), 6 (illegal op code), and on 7 (x87 not available). On all machines it takes over **int 9** (keyboard), **20h** (return to DOS), **21h** (main MSDOS software interrupt), **22h** (terminate address), **23h** (Ctrl-Break), **24h** (Critical Error Handler), and **27h** (return resident to DOS).

SST takes over the keyboard input interrupt 9 to see if a Ctrl-Enter has been typed. If so, SST takes control, which allows you to stop a runaway program. For any other key combination, SST transfers control to the keyboard routine active at the time SST was loaded. SST takes over the MS-DOS interrupt 21h to intercept DOS exit requests (**ah**=0, 31h, and 4Ch). If these are encountered, SST takes control issuing the message "Program terminated normally." If not, SST transfers control to the MSDOS program active at the time SST was loaded.

3-6. Hardware Interrupts

For hardware interrupts, the corresponding interrupt programs can be invoked either by a real hardware interrupt or by software executing an **int**, far **call**, far **ret**, **iret**, or far **jmp** instruction. SST identifies the software **int** *n* cases as such, and otherwise gives the appropriate hardware interrupt message. For example, if your program executes an **int 0** instruction, you'll see the message int 0, rather than code leading to the interrupt using the **u**nassemble command.

Another example on the IBM PC is the message "PARITY CHECK 2", which allegedly means that some memory location may have caused the error. You can run a memory test program to check your memory, but the interrupt could have been caused by a software bug, namely an **int**, **call**, **ret**, or **jmp** that uses interrupt vector 2, the nonmaskable interrupt vector. Running under SST, you can check the origin of the problem, and either go fix your program or your memory accordingly.

NMI Button

Very usefully, the NMI interrupt can be caused by your shorting the I/O Channel CHK line to ground with an NMI button (connect normally open push button switch to top and bottom I/O Channel pins closest to rear of PC). This is a very powerful way of giving SST control when ordinary maskable interrupts have been disabled.

Interrupt Mask Control

Particularly in debugging multitasking systems that use the system clock to switch tasks, it is important that SST can control the interrupt mask active when

SST is active. Otherwise, SST could regain control and immediately lose it to some other task. For this purpose, the command

▷qi *mask*

sets the interrupt controller mask used when SST is active to the value *mask*. For example on the IBM PC, to allow only keyboard interrupts, use

▷qiFD

You can try this option out with the clock option. This shows the seconds ticking away in the upper right corner of the screen. When the 0FDh interrupt mask is used, SST stops the screen update whenever SST is active, and then restarts it upon return to the user program.

3-7. DOS and x87-Emulation Interrupt Definitions

SST automatically comments some unassembled instructions, such as DOS calls (**int 21h**) and the x87 emulation interrupts (**int 34h - int 3dh**). In addition the **int 21h** definitions are displayed when you type the int21[*n*] command in COMMAND MODE. If the optional *n* is present, the definition for that entry point alone is displayed. If *n* is missing the next hexadecade of **int21** entries is displayed. These features are very handy for working with code that makes DOS calls.

Examining Interrupt Vectors

To facilitate examining interrupt vectors, the suffix "i" on an address *address* automatically implies the address 0:4**address*. Hence the command

▷dd17i

displays the real-mode interrupt vector table with the cursor position at 0:5Ch, which has the double-word vector to **int 17h**. You can then type Ctrl-F to begin disassembling at this routine.

4. SST Syntax, Labels, and Strings

The preceeding chapters show lots of SST examples, so you probably already have a pretty good idea of what the syntax of various SST commands is as well as the syntax we use in this manual. The present chapter summarizes the nature of this syntax more completely, describes executing DOS commands from within SST, and explains how you can configure the keyboard editing commands to correspond to your preferences.

4-1. Syntax

SST accepts commands in the same format as DEBUG.COM, so users of DEBUG can continue with their usual methods. Most SYMDEB.EXE commands are also supported with the same notation. In addition, many DOS or BASIC like commands are supported and the two kinds of commands live together remarkably peacefully. The DEBUG-style alphabetic commands are identified by a single command letter that can be preceded or followed by optional blanks and tabs. Most command take one or more arguments separated by a blank, comma, or tab. The syntax has been relaxed in several ways to streamline command entry and execution. The semicolon (;) can be used in place of the colon (:) for specifying segment register values. As in DEBUG, the segment register names (**cs**, **ds**, **es**, and **ss**) can be used to specify address segments. SST allows the program registers (**ax**, **bx**, **cx**, **dx**, **al**, **ah**, **bl**, **bh**, **cl**, **ch**, **dl**, **dh**, **si**, **di**, **bp**, **sp**, **ip**, **eax**, **ebx**, **ecx**, **edx**, **esi**, **edi**, **esp**, **ebp**, **eip**, **fs**, and **gs**.) to be used as well in place of hexadecimal values. Five-digit addresses refer to the entire one-megabyte address space, and 6-digit address correspond to extended RAM available on the IBM PC compatible computers with 80286 and later microprocessors. If no segment register is given, **ds** is assumed for all commands except for **a**ssemble, **g**o, **l**oad, **t**race, **u**nassemble, and **w**rite commands, which assume **cs**.

Syntax Type Fonts

In this document we print register and instructions in **boldface Times Roman**, unless they appear in examples, which are displayed in an Arial font. Each

command is defined and described with examples, and is compared to its DEBUG version. The Enter key is indicated by ↵, and is used to terminate a command line, and to terminate the execution of certain commands like the DISPLAY and TRACE MODEs. The usual SST prompt character is shown as ▷ (similar to the SST prompt on the IBM PC screens.) Variables are given in *italics*.

Each command is defined by a SYNTAX specification. In these specifications, bold square brackets [ ] are used to surround optional fields. For example,

▷a [*address*]

means that the letter a is typed following the SST prompt ▷, optionally followed by the address *address*. The text following the syntax specification then defines what a alone means and what a followed by an address means.

In the syntax specifications, the word *range* stands for two addresses separated by a comma or blank. The first address can have an optional segment specification. The segment used for this first address is automatically used for the second (the second address for the protect command can have its own segment). The range can be used in commands like display, fill, and compare. For example,

▷d [*range*]

displays the range of memory *range*. This range can be given in one of three forms:

1. $address_1$ $address_2$
2. $address_1$ L *count*
3. Ctrl-B

The first form specifies the address explicitly either with hexadecimal values of the form [*segment:*] *offset*, or with labels (see Label section in this Chapter). The second form specifies the number of bytes including the first one pointed to by the address $address_1$. The third method uses a block defined by the Ctrl-T and Ctrl-E DISPLAY MODE options (see Block section in this chapter).

If during execution you type Function Key 1 anywhere after the command letter, the short syntax definition is displayed in a pop-up window under the register window. Type any character to get rid of this help screen. If you type Function Key 1 immediately after the command letter, both are erased when the help screen goes away. If you have typed more than two characters when you type Function Key 1, then those characters are left, letting you continue your command after seeing the help screen. The assemble command works a bit differently in having two help screens. The first time you type Function Key 1 a terse syntax

definition is displayed, followed by the full set of possible x86 instruction mnemonics. Typing Function Key 1 again toggles the Assemble help screen to the x87 mnemonics. Typing any other character replaces the screen text and cursor to that present before you asked for help. This lets you check the spelling of a mnemonic in the middle of typing an instruction.

Specifying an Argument Using Cursor Arrow Keys

If the desired hexadecimal value for an argument (usually an address) appears on the screen due to displaying or unassembling, you can insert this value into your command line. Use the cursor arrow keys to move the cursor to the start of the value and then type a blank to continue the command or a ↵ to run it. Both the address at the cursor as well as the character you type are inserted into the command string. This is particularly useful for beginning a trace at a disassembled instruction displayed on the screen, for setting a breakpoint at a disassembled instruction, or for starting a disassembly at an address found by searching for instruction mnemonics. For example, to start execution at the current instruction (**cs:ip**) and break at an address displayed on screen, type

```
▷g
```

with no ↵, move the cursor to the desired address on screen, and type ↵. If you want a second breakpoint, press the space bar instead of the ↵, move the cursor to the second address, and then type the ↵. If you just want to copy in the segment value from one part of the screen, move the cursor to the start of the segment value and type a backspace. This inserts the segment, the colon and only three digits of the address offset, three more backspaces delete the rest of the offset to make room for the value you want. If you're an accurate typist, inserting a segment value this way isn't that useful, but it illustrates how whatever you insert this way can be further edited, unless you type the terminating ↵.

You can use the arrow-key insertion method to assemble on top of code, or to set a single temporary breakpoint. You can also do these things directly from the UNASSEMBLE and TRACE MODEs.

4-2. MS-DOS Commands

A subset of MS-DOS commands has been built into SST allowing you to change the default drive and directory path, to display directory filenames, to type files, and to erase files. The display runs about three times as fast as MS-DOS, and the type command at least 20 times as fast. Hence the type command runs in a special paged mode.

To change the current MS-DOS drive letter, type the desired drive letter followed by the command colon (or semicolon). For example,

▷a:

switches to drive A and displays a screen showing the current drive (A) and directory path on that drive.

To change the default directory path on the current drive type cd\\ followed by the path name. Hence

▷cd\sst

changes to the subdirectory \SST, and displays a screen to that effect. If no such directory exists, the screen shows the active directory instead.

To display the current default drive and subdirectory, type \ Function Key 1 in COMMAND MODE or type the prompt command in COMMAND MODE to continually see this information.

To display **dir**ectory information, type dir followed by the desired filename template. The total number of bytes in the files matched is displayed in decimal for all values if you have an x87, and up to 64K if you don't (larger values are then displayed in hex).

To display a file, type

▷type *filename*

which allows you to browse up and down a file with search capability.

To erase a file, type

▷erase *filename*

or

▷del *filename*

You will be asked to confirm before SST erases the file.

To leave SST, you can type bye, quit, or system, in upper or lower case. For a complete summary of these commands, see "Interpreter Commands" in Chap. 6.

4-3. Editing Command Lines

SST has a line edit facility patterned after the PMATE editor that works both in COMMAND and ASSEMBLE MODEs. While typing in a command or assembly language statement, you can use the left and right arrow keys to move around the

line. Typing ordinary characters simply inserts them at the cursor position. Other edit keystrokes are identified as follows:

Home	move cursor to beginning of line
End	move cursor to end of line
Del	deletes character under cursor
Backspace	deletes character before cursor
Ctrl-O	move one word left (Ctrl-← is an alias)
Ctrl-P	move one word right (Ctrl-→ is an alias)
Ctrl-Q	delete word to left
Ctrl-W	delete word to right
Ctrl-K	delete (Kill) from cursor to end of line

Modifying Edit Command Characters

If you prefer, you can load in a file to reconfigure the Ctrl character commands. For example, to change the commands to correspond to MicroPro's WordStar editor, type

```
key 1e01, 1f13, 2004, 2106, 0, 2308, 2207, 1414, 1519
```

Each entry specifies the desired IBM PC character code for an edit function. These functions appear according to the order:

WordLeft,	CharLeft,	CharRight,	WordRight
DeleteWordLeft,	DeleteCharLeft,	DeleteCursorChar	
DeleteWordRight,	DeleteToEndOfLine		

Hence in the WordStar example above, the first entry 1e01 (Ctrl-A) corresponds to moving left one word, while the second entry 1f13 (Ctrl-S) corresponds to moving left one character. If you're used to the WS commands, include this key command in your SST.INI file.

Blocks

It's often useful to scan through memory using the DISPLAY MODE and then examine part of memory in a different way, or write it out to disk. For such purposes, SST has a limited form of the word processor block facility. Specifically any time you type Ctrl-T in DISPLAY MODE, the address pointed to by the cursor is saved in a double-word Tag location.

You can examine the memory at the Tag location using the examine command of Chap. 5 by typing Ctrl-B instead of an address. The Ctrl-B so used displays as a little box.

Similarly if you type Ctrl-E in DISPLAY MODE, the address pointed to by the cursor is saved in a double-word End tag location. You can go back and forth between these two locations by typing Ctrl-G.

You can use the block of memory between the Tag and End-tag locations in a number of ways. You can write a block to disk by naming the desired file with the **n**ame command and then typing

▷w Ctrl-B

in COMMAND MODE. The Ctrl-B can also be used in place of two addresses with the compare, **f**ill, **m**ove, and search commands. For example,

▷f Ctrl-B "abcd"

fills the memory defined by the previous Ctrl-T Ctrl-E DISPLAY MODE option with the string "abcd". The **c**ompare, **f**ill, **m**ove, and search options are limited to 64K, but the **w**rite option is limited only by the RAM and disk sizes.

4-4. Labels

The load map generated by the DOS LINK.EXE program with the /MAP list option can be loaded by SST to identify locations in memory by name. This works with large and small memory model programs and greatly facilitates debugging programs. Once defined, the labels can be used in place of hexadecimal addresses. For example to start unassembling at the double-word address alpha, type

▷u alpha

To **l**oad in program **l**abels, name the .MAP file with the **n**ame command (Chap. 5), and type the ll command (see Chap. 5 **l**oad command). This automatically reads the labels in starting at the point in the .MAP file identified by the words "by Value" and relocates them relative to the origin of the .EXE module (program prefix segment paragraph + 10).

To **l**oad .MAP **l**abels relative to some other paragraph, type a command of the form ll *n*, where *n* is the desired paragraph number. This option is useful for debugging resident programs. To **l**oad .MAP files for use with .COM files, type lm, which relocates relative to the Program Segment Prefix (PSP) rather than to the .EXE module paragraph (10h paragraphs lower)

SST has limited support for program variables with the lv option. This option loads the variables defined by the part of a MASM.EXE listing for a single segment. The program scans for the word "segment" and sets up program variable names up to the corresponding **ends** pseudo op.

The ll, lm, and lv options use the user program area to load in the .MAP and .LST files and hence overwrite whatever program might have been loaded in. Hence to debug a program, load in the label files first, and then the program. Using a script file (see < command) streamlines this procedure. Labels can be displayed by segment paragraph number by the d/*n* command of Chap. 5. Variable names are displayed by d/v and user strings by d/u.

The LL option can also read in label files generated by the wl option, which writes the entire SST label linked list to disk, program labels, variable names, and user strings. See the **w**rite command in Chap. 5.

For the more technically oriented, we note that internally SST stores all labels in a two-level linked list format. The outer level linked list consists of one or more entries having a segment paragraph value (one word), followed by a word type code, a word segment label string length, and a string of that length. In turn, an outer level string contains one or more inner level linked lists, each describing a label. An inner level linked list has the same format for its entries, with a one word offset value, a word type code, a word string length, and a label string of that length. The special segments for lv and lw are identified by the outer level pseudo-segment numbers 0FF00H+"V" and 0FF00H+"W", respectively, which are not likely to occur as program paragraphs. When you use the wl option, a double linked list of this form preceded by the special word 0FFFAh is written to the file named by the **n**ame command. In writing the file, the paragraph values less than 0FF00h are unrelocated by subtracting the Program Segment Prefix paragraph + 10h. This file can subsequently be reread by the ll option, which adds the Program Segment Prefix + 10h paragraph back in. In this way, the labels can be used when your program is loaded in a different place on subsequent occasions. Pseudo segments 0ff80h and up are used for virtual segments in debugging Microsoft Windows.

4-5. User Strings and Keyboard Macros

User strings are definable with two character alphanumeric names. These strings can be used for defining memory structures or *templates* for use with the **e**xamine memory command (see Chap. 5), and for user input (macros without arguments). The facility makes it much easier to read the values of data structures stored in memory. To define any user string, type

```
▷nst="..."
```

This defines (names) the string st to have the value ... .

You can also define 40 function key values by assigning strings to f0 - f9, c0 - c9, s0 - s9, and a0 - a9, which define the unshifted, Ctrl'd, Shifted, and Alt'ed functions, repectively. For example,

```
▷f9="d
dd;
"
```

defines Funtion Key 9 to switch back to COMMAND MODE if it's not there already, and to display the real-mode interrupt vectors in double-word format.

To use a string in place of keyboard input (limited macro facility), type the command $ followed by the string name. Hence the sequence

```
▷nst="dd;"
▷$st
```

generates a full screen double-word display of the real-mode interrupt vectors, just as if you had entered the command dd; directly.

Alternatively, you can put these commands in a file and run them using keyboard redirection. This method is really simpler. For example we might debug SST.EXE by using the file S

```
nsst.map
ll
nsst.lst
lv
nsst.exe
l
qs3
```

and then invoke this file by the keyboard redirection command

```
▷n<s
```

or simply

```
▷<s
```

5. Command Descriptions

This Chapter describes the basic SST commands summarized under Chap. 2 on Help as

!&.<>	0-9	@scii	Asm	Baud	Comp	Display
Exam	Fill	Go	Hex	In	Klear	Load
Move	Name	Out	Protect	Quit	Reg	Search
Trace	Unasm	Vector	Write	Xam	YGDT	Zam

It addition, this chapter summarizes many MS-DOS-like commands as given in the Table of Contents. Commands specific to the interpreter and the disk editor are described further in Chaps. 6 and 7, respectively. The commands are introduced by a brief syntax specification (see Chap. 4) followed by an explanation of their usage and some examples.

! SHELL Command

The command ! loads and executes copy of the DOS COMMAND.COM. The syntax is

▷! [*filename* [*parameters*]]

If the optional *filename* and *parameters* field are present, the file is executed with the command-line parameters specified and control is returned to SST. If the ! appears alone, COMMAND.COM retains control giving the user the usual DOS command line prompt. Type any commands desired and return to SST by typing the DOS command exit.

Note that SST and whatever you're debugging remain resident during this process, substantially reducing the amount of RAM left to the new COMMAND.COM process, relative to the one used to run SST in the first place. Many DOS commands such as dir are built into SST, so in many cases you may not have to use the shell command.

& Address Command

The & *label* command returns the address of the variable or label *label*. If *label* is not in the symbol tables read in (see LL command), an error message is issued. Type the d/s command to see what segments have symbol tables.

0-9 Calculator Command

Commands beginning with 0-9 invoke the 32-bit hex calculator option as described in Sec. 3-4.

< Command

The < *filename* command redirects keyboard input to the file *filename*. This is very useful for reading in script files to define symbol tables, function keys, and initialization commands.

> Command

The > *filename* command defines the file *filename* to be used for echoed output.

@ Command

The @scii command displays hexadecimal and decimal ASCII charts as described in Sec. 2-2.

A Command

The a command assembles x86 and x87 mnemonics. It has the syntax:

▷a [*address*]

This starts assembling instructions typed in by the user at the address given following the a, or at the last address used (initially cs:100) if no address is given. The menu line (third line from screen top) changes to

ASSEMBLE MODE: F1 Esc

The help screens displayed by Function Key 1 are discussed below. The first time an a ↵ is typed you see (suppose cs=1234)

```
▷a
1234:100 _
```

After you terminate an assembly language instruction with a ↵, the screen displays the corresponding machine language and goes on to the next line. The assembly language mode is terminated by two ↵'s in a row. Thus to add the first ten integers one types and sees

```
▷a
1234:100 B90A00  mov   cx,a
1234:103 31C0    xor   ax,ax
1234:105 01C8    add   ax,cx
1234:107 E2FC    loop  105
1234:109
▷
```

You can screen trace the operation of your program by typing t or T followed by ↵ and single stepping by typing the space bar. From the TRACE and UNASSEMBLE MODEs, you can invoke the assembler on the line given by the cursor by typing the hot key "a".

Assembler Syntax

In the ASSEMBLER MODEs, all numbers are assumed to be hexadecimal unless identified as decimal by a trailing period, or identified as a string literal by enclosing in single or double quotes. Extra spaces and tabs can be be freely inserted to improve readability. Memory references are usually chosen to be byte or word according to the register that appears in the instruction. Hence the instruction mov [100],ax moves a word to the location 100, while mov [100],al moves a byte. If no register is mentioned, such as in immediate transfers like mov word ptr [100],10, the usual modifiers word ptr and byte ptr can be used. These can be abbreviated by word and byte, respectively, or simply by w, and b,. For example,

```
shl   word ptr [100],1
shl   word [100],1
shl   word [100],1
shl   w,[100],1
```

all assemble the machine instruction that shifts the word at location **ds**:100 left one bit position. Either a byte or a word specification must be given. In general syntax accepted by DEBUG.COM is accepted by SST as well.

In ASSEMBLE MODE, Function Key 3 displays code for instruction at current address for editing. You can also edit a sequence of instructions in a row by using the edit command described in Chap. 6.

x87 Instructions

For the x87 instructions, the specifications d,, q,, and t, are available to mean double word, quad word, and ten byte (temporary real — **fld** and **fstp** instructions only), respectively. The usual assembler notation dword, qword, and tbyte followed optionally by ptr is also accepted. The stack registers can be referenced by their full names st(*i*) with *i*=0 to 7. For simplicity they can also be referenced by st0 to st7, with st alone meaning st(0). Most simply, they can be referenced by 0 to 7, an unambiguous specification since no immediate instructions exist on the x87.

The arithmetic instructions **fadd**, **fmul**, **fsub**, and **fdiv** can appear with no arguments, in which case the operand field st(1),st(0) is implied and a pop occurs. For example, **fadd** means

```
faddp st(1),st(0)
```

This abbreviation makes arithmetic instructions without operands work as in a Polish suffix calculator.

x86 Mnemonics

To see all the mnemonics recognized by the assembler, type the command a F1. This displays the help screen

a [address] Assemble. x86 op codes (8086 lower case, F1 → x87)

aaa	aad	aam	aas	adc	add	and	Arpl	Bound	Bsf
Bsr	Bswap	Bt	Btc	Btr	Bts	call	cbw	Cdq	clc
cld	cli	Clts	cmc	cmp	cmpsb	Cmpsd	cmpsw	Cmpxchg	Cmpxchg8
Cpuid	cwd	Cwde	daa	das	db	dec	div	dw	end
Enter	esc	hlt	idiv	imul	in	inc	Insb	Insd	Insw
int	into	Invd	Invlpg	iret	Iretd	ja	jae	jb	jbe
jc	jcxz	je	jecxz	jg	jge	jl	jle	jmp	jmps
jna	jnb	jnc	jne	jng	jnl	jno	jnp	jns	jnz
jo	jp	jpe	jpo	js	jz	lahf	Lar	lds	lea

Leave	les	Lfs	Lgdt	Lgs	Lidt	Lldt	Lmsw	Loadall	lock
lodsb	lodsd	lodsw	loop	loope	loopne	loopnz	loopz	Lsl	Lss
Ltr	mov	movsb	Movsd	movsw	Movsx	Movzx	mul	neg	nop
not	or	out	Outsb	Outsd	Outsw	pop	Popa	Popad	popf
Popfd	push	Pusha	Pushad	pushf	Pushfd	rcl	rcr	Rdmsr	Rdtsc
rep	repe	repne	repnz	repz	ret	rol	ror	Rsm	sahf
sal	sar	sbb	scasb	scasd	scasw	seg	Set	Sgdt	shl
Shld	shr	Shrd	Sidt	Sldt	Smsw	stc	std	sti	stosb
Stosd	stosw	Str	sub	test	Verr	Verw	wait	Wbinvd	Wrmsr
Xadd	xchg	xlat	xor						

Typing any character other than Function Key 1, restores what was on the screen before you asked for help. If you type Function Key 1 again, you see the x87 mnemonics:

x87 Mnemonics

a [address] Assemble. x87 op codes (F1 ® x86) are:

f2xm1	fabs	fadd	faddp	fbld	fbstp	fchs	fclex	fcom	fcomp
fcompp	Fcos	fdecstp	fdisi	fdiv	fdivp	fdivr	fdivrp	feni	ffree
fiadd	ficom	ficomp	fidiv	fidivr	fild	fimul	fincstp	finit	fist
fistp	fisub	fisubr	fld	fld1	fldcw	fldenv	fldl2e	fldl2t	fldlg2
fldln2	fldpi	fldz	fmul	fmulp	fnop	fpatan	fprem	Fprem1	fptan
frndint	frstor	fsave	fscale	Fsetpm	Fsin	Fsincos	fsqrt	fst	fstcw
fstenv	fstp	fstsw	fsub	fsubp	fsubr	fsubrp	ftst	Fucom	Fucomp
Fucompp	fwait	fxam	fxch	fxtract	fyl2x	fyl2xp1			

For use of these commands, see one of the Intel x86 Programmer's Reference Manuals, the Microsoft *Macro Assembler* manual, or the book *The Personal Computer from the Inside Out*, 3rd Ed (1994) by Sargent and Shoemaker.

If an error is found, the **a**ssemble command indicates the offending letter or field by an Up arrow followed by the word error as in

```
mov   ax,q
      ↑ error
```

You can then use the command edit keys (see Chap. 4) to fix the error, or type Esc to return to COMMAND MODE.

Labels and Comments

You can label instructions as you type them in with the **a**ssemble command. Terminate the labels with a colon (:), and they are inserted into the label table for

use in later assembles, unassembles, examines, displays, etc. A colon alone deletes the label at the current program counter. A new label replaces an old one. You can save the labels you type and/or read in with the wl command. Use the ll command to read in labels written with the wl command.

You can also define labels by reading them in from the /MAP option (ll command) on the linker and /LST option (lv command) on the macroassembler. The assembly language interpreter (Chap. 6) can have variables of its own by using the db and dw pseudo-ops, or by storing a value into an as-yet undefined variable.

Program comments are supported by the assemble and unassemble commands and are defined by a starting semicolon. If you type a semicolon at the start of a line, the comment that follows will automatically be appended to the instruction for that line. If you type a semicolon alone, any existing comment for the line is deleted. If you type a new comment, it replaces the old one.

Program Mode

The end pseudo op stops TRACE MODE unassembly beyond the end, giving a more readable screen. This pseudo op can only be used in .COM file mode, i.e., with cs equal to the program prefix segment. When end is in effect, you can edit, insert, and delete instructions in the middle of your program. See Chap. 6 on the built-in assembly language interpreter.

The a option acts essentially the same way as for DEBUG.COM with the additions of instant help messages (just type Function Key 1), of displaying the assembled machine language, of recognizing the x86 mnemonics beyond the 8086, use in search strings, with the change to lower case for increased readability, use of labels and comments, and with the PROGRAM MODE allowing insertion and deletion of instructions.

SST also uses the assembler for the Super-Trace and conditional breakpoint facilities.

AND Command

The logic operations and, or, xor, and not operate on memory ranges much like the fill command. The and, or, and xor have the same syntax as the fill command, while the not command simply inverts all bits in the range. The syntax is

▷and *range list*
▷or *range list*
▷xor *range list*
▷not *range*

The bytes in the list *list* are **and**ed, etc., with the bytes in the range *range*. The and operation can be used to kill the high bit on bytes in WordStar files or the parity bit included by some communications programs. For example, to kill the high bit in the file WORDSTAR.DOC, type

```
▷nwordstar.doc
▷l
▷and 100 l cx 7F
▷w
```

B Command

This command manages breakpoints and setting the **b**aud rate. To set the baud rate, type a command of the form

▷b *rate* [,*channel*]

where *rate* = one of 110, 150, 300, 600, 1200, 2400, 4800, 9600. The optional channel value of 1 specifies COM1 and 2 gives COM2.

If no number is given, the message

Reboot System (Y/N)?

is displayed. Typing y or Y reboots the system without erasing memory (works only on older IBM PC's).

Breakpoint Commands

In addition to the 10 normal **g** breakpoints (see **g**o command) that go away upon reentry to SST, you can define up to 10 sticky breakpoints with the **b**reakpoint command. To set sticky **b**reakpoints type a command of the form

▷bs[*n*] *address* [*m*]

which sets breakpoint [*n*] at the address *address* and skips *m* passes by this address. No blanks can occur between *n* and the s if *n* is given.

To **c**lear **b**reakpoints in *list* or all (*), type a command of the form

▷bc *list*

or type

▷bc *

The list *list* here refers to one or more digits 0 to 9, and the * refers to all ten possible breakpoints.

Similarly to **d**isable **b**reakpoints in *list* or all (*), type a command of the form

▷bd *list*

or type

▷bd *

and to **e**nable **b**reakpoints in *list* or all (*), type a command of the form

▷be *list*

or type

▷be *

To **l**ist your **b**reakpoints and their characteristics, type

▷bl

For example,

▷bs0 1234:5678

defines and enables sticky breakpoint 0 at the address 1234:5678.

▷bd0

disables sticky breakpoint 0. Hence if you **g**o to the program, this breakpoint will not be set. The be0 command can reenable the breakpoint.

After defining one or more sticky breakpoints, type the bl command to see a pop up window telling you about the status and location of the breakpoints. This facility is compatible with SYMDEB.EXE's except that the latter uses **bp** instead of bs to set BreakPoints. SST uses **bp** to refer the **bp** register.

80386 hardware breakpoints

The 80386 and later microprocessors have four hardware breakpoints capable of breaking when a location (byte, word, or dword) is read, written, or exe-

cuted. These breakpoints occur while the 80386 runs full speed. In particular, you can break execution as soon as a variable is written to or from, and you can set breakpoints in ROM. For a full discussion of this powerful feature of the 80386, please consult the chapter on debugging in the Intel *80386 Programmer's Reference Manual* or one of its more recent siblings.

SST uses sticky breakpoints 30 to 33 are for the 80386 debug registers. The syntax is a slight extension of the standard sticky breakpoint syntax, namely

▷bs *n address* [*m*] [*/stl*]

which sets 80386 hardware breakpoint *n* = 30 to 33 at the address *address* and skips *m* passes by this address. The special 80386 optional switch field */stl* can be used to specify the scope, type, and length of hardware breakpoint desired. The scope letter *s* can be omitted or be g for Global, or l for Local. Global is the default. If you specify Local, the breakpoints are automatically disabled when a task switch occurs. *l* = 1, 2, or 4 (bytes).

The breakpoint type letter *t* can be r for break on data reads or writes, but not on instruction fetches; w for break on data writes only (default); or x for break on instruction execution only. These 80386 hardware sticky breakpoints can be enabled, disabled, cleared, and listed just as regular sticky breakpoints can.

BLINK Command

The command

▷blink [*n*]

controls the Hercules Graphics Card Plus blink/reverse-video attribute. If *n* = 0 or is missing, then bit 7 = 1 of the screen attribute byte specifies a high-intensity background. If *n* = 1, then bit 7 = 1 of the screen attribute byte specifies blinking. Use blink 0!

BYE Command

The command

▷bye

quits to DOS. system, quit, and Alt-X do the same thing.

BYTE Command

To facilitate both source-level and assembly language debugging, SST includes commands to define typical data types. The syntax for the byte type is:

▷byte [[far] *] *variable-name address*

For example,

▷byte alpha 305

adds the symbol alpha of type byte (8-bit unsigned integer) to the segment specified by the **ds** segment register at the offset 305h. You can specify any other segment. The optional * generates a near ptr to a variable of the type byte. The optional far generates a far pointer to a variable of the type byte.

C Command

This command compares the contents of one memory block to that of another memory block (like DEBUG). It is useful for checking that two copies of a program are identical and have not been changed, for example, by a crash of the system. The command expects the first block start and end addresses and the second block start address as arguments. Syntax:

▷c *range address*

For example,

▷c500,595,2000

compares the contents of memory from address 500h through 595h against the contents of memory from address 2000h through 2095h. Any differences will be shown on the display. Thus if location 527h was 00 while 2027h was FFh in the above example, the display would show

▷c500,595,2000
2347:527 00 FF

assuming all other locations in the blocks are identical.

If **ip**=235, **cs**=0200, **es**=3000 and **di**=495, then

▷ccs:ip E000 es:di

compares 0200:235 up to 0200:E000 against the memory block starting at 3000:495.

This command works the same way in DEBUG, and allows general register names to be used in the address fields.

CD Command

The cd command changes the directory as for DOS. Such changes cause the new path name to appear in a pop-up window. Alternatively, type the prompt command (described later in this chapter) to display the current path on all COMMAND MODE command lines (as for the DOS prompt command). The instead of the ▷ prompt, you see something like

```
C:\PS>
```

To change the current drive, type a command of the form *a*: or *a*;, where *a* is the desired drive letter.

CHAR Command

To facilitate both source-level and assembly language debugging, SST includes commands to define typical data types. The syntax for the char type is:

▷char [[far] *] *variable name address*

For example,

▷char alpha 305

adds the symbol alpha of type char (8-bit signed integer) to the segment specified by the **ds** segment register at the offset 305h. You can specify any other segment. The optional * generates a near ptr to a variable of the type char. The optional far generates a far pointer to a variable of the type char.

CLOCK command

SST can display a time-of-day clock at right side of menu once a second. The command

▷clock [*status*]

turns this clock on or off if *status* = on or off, respectively. If *status* equals an address instead of on or off, the word value at that address is displayed once a second. In addition to reporting the time of day, this feature lets you know if your machine has totally crashed.

CLOSE Command

The close command closes all opened files except those with handles 0 through 4.

CLS Command

The cls command acts as for DOS to clear the screen, here leaving the register window on top.

CONFIRM Command

The confirm command is typically used in demonstration script files to bypass the need for the user typing "y" or "n" to confirm an action. The syntax is

▷confirm *status*

where *status* = off turns off the need to confirm, while the value on turns it back on (default is on).

CONT Command

The cont command continues operation where left off. It is the same as the g command without arguments, except that cont requires no confirmation if no breakpoints are implemented.

CPU Command

The cpu command gives you information about your computer. For example when you type

▷cpu

on a typical PC, you might see

```
CPU: 80486/no 80487
RAM: 0 1 2 3 4 5 6 7 8 9 A B C D E F
Serial Ports: 3F8 2F8
Parallel Ports: 3BC 378 278
Model: fc-1
```

```
DOS: 6.22
Speed Relative to PC I: 51.7
BIOS ROM: 01/13/93
DPMI: 0.90
Windows: 3.11
```

Here the underlined values in the RAM entry are shown in reverse video on screen and signify that RAM exists in those 32K memory banks.

The program used to measure the speed relative to the IBM PC I is

```
;SPEED - display null-terminated string ds:[si] followed by ratio of loop speed on
;machine to that of IBM PC I.  ax, bx, cx, dx, si changed

speed:  cli             ;No interrupts
        call    mark    ;Get ax = timer start count
        mov     cx,800h
speed2: push    ax      ;Delay
        pop     bx      ;bx = original timer count
        loop    speed2
        call    mark
        sti             ;Interrupts back on
        sub     bx,ax   ;bx = binary interval count
        call    tom     ;Display speed message
        mov     ax,0c100h ;IBM PC I loop time (0c100)
        xor     dx,dx
        div     bx      ;ax = integer ratio
        push    dx      ;Save remainder in 0f000's
        call    dlbyte  ;Display integer part
        mov     al,"."  ;Display decimal point
        call    co
        pop     ax      ;ax = remainder
        mov     cx,10d  ;Convert to tenths
        mul     cx
        div     bx
        jmp     dlbyte  ;Display tenths' part and ret
;MARK - return ax = timer 0 count.  ax changed

timer0  =       40h     ;8253 timer 0
timctl  =       43h     ;8253 timer control port

mark:   mov     al,0    ;Latch count
        out     timctl,al
```

```
in      al,timer0
mov     ah,al
in      al,timer0
xchg    al,ah
ret
```

CSRSIZE Command

The command

▷csrsize *xxyy*

begins the cursor on character raster row *xx* (starting with row 0 on the top) and ends the cursor on row *yy*. Hence the command

▷csrsize b0c

gives a two-line cursor starting on line 11 (decimal) and ending on line 12.

D Command

This command **d**isplays the contents of memory in hex/ASCII (like DEBUG), pure ASCII, word, or double-word formats. In the hex/ASCII mode, sixteen bytes of memory are displayed per line with the starting address of the line given as the first entry on the line. In the pure ASCII mode, 64 (40 hex) bytes are displayed per line, allowing one to see four times as much memory on screen as with the hex/ASCII mode.

The command

▷d [*range*]

displays the memory in the range *range*. For example,

▷d100,200

displays memory from address 100h to address 200h inclusive. Usually this syntax is compatible with DEBUG.COM. However if one of the characters "abdpw" follows the d *with no intervening blanks*, that character is interpreted to choose the DISPLAY MODE ASCII, BYTE (hex/ASCII), Double word, triple-nibble (for 12-bit FAT displays), and Word, respectively. The mode so chosen is used by subsequent **d**isplay commands until overruled. A particularly handy special case is the command

▷dd;

which displays the interrupt vectors down at 0000:0000 in double word format. You can then use the cursor keys, blanks and backspaces to move to the desired interrupt vector and type Ctrl-F to start unassembling at the interrupt handler entry point. Note that **dd** *ni* displays the interrupt vector table with the cursor at interrupt vector *n*.

The output of this start/end **d**isplay option can be written to a file of your choice (see n> *filename* command). Under MS-DOS 2.0 and later versions, this file can be the printer instead of a disk file. The **d**isplay command for a *range* quits with Ctrl-C and pauses with Ctrl-S. The maximum range length is 8000h.

Linear Address Display

A linear address display is available for Real Mode and protected 386 mode operation. In COMMAND MODE, type

▷dx *offset*

where on an 80386 the offset can be 32-bit. Using this command in protected mode on ISA-bus PCs, you can see that this machine wraps its address space at 16 megabytes. If an address greater than 1 megabyte (100000H) is used, the "x" is implied. This display mode is very handy for looking beyond 1 megabyte.

Screen Display

If no address or only the start address is specified, an instantaneous screenful of memory is displayed with a register/menu window on top. The offset at the cursor is displayed in the register window and the registers are displayed as discussed under the **r**egister command below. SST has the menu

DISPLAY MODE: ←↑↓→ F1 Tab ASCII Tag Word Dbl Near Far Cont Ovr Undo

The Tab entry means the Tab key. The A stands for Ctrl-A. To type this, type a or A while holding the Ctrl key down. This toggles the DISPLAY MODE between hex/ASCII and pure ASCII formats (see below). Typing Function Key 1 displays the help screen (Ctrl characters are shown in reverse video on screen)

←↑↓→	move cursor (csr)	PgUp/Dn	scroll screen
Tab	csr: HEX ↔ ASCII	ASCII	toggle ASCII/HEX
Byte	Byte format	Shift	invert case at csr
Tag	Tag csr position	End	End block at csr
Go	tag ↔ end	Restore	Restore csr
Z	display last char read	@scii	ASCII screen
Ovr	toggle OVERTYPE	Undo	Undo last overtype
Word	csr = Word ptr	Double	csr = Dword ptr
Near	csr = Near unasm ptr	Far	csr = Far unasm ptr
Cont	tgl Continuous update	Enter	go to COMMAND MODE
V	unassemble at csr		

<ctrl>Y =DISPLAY EBCDIC (see pg 13)

Cursor Movement and Memory Display Format

The cursor arrows, PgUp, PgDn, space bar, and backspace move the cursor around memory, scrolling the screen to keep the cursor on the middle line of the display (except near 0000:0000). Memory can be displayed in two formats, hex/ASCII and pure ASCII. The hex/ASCII format displays 16 (10h) bytes of memory per line, in hexadecimal form on the left and middle of the screen, and in ASCII on the right, as shown on the following page. The ASCII column replaces control characters (those with codes less than 20h) by periods. The pure ASCII format displays 40h bytes per line, with all control characters except code 0 by periods. Code 0 shows up so often that SST represents it by a ∘ character.

Vertical motions can extend arbitrarily far up or down. In the pure ASCII mode, the autorepeated PgUp command displays each 64K RAM of memory in about 4 seconds, making it easy to scan all of memory for text. If you scroll towards lower addresses in memory than those displayed at the start of the display command, the address segments decrease by 10h. This allows you for example, to examine the bytes in the program prefix of an .EXE file with the program prefix segment displayed. If you scroll up in memory beyond the segment you started with, the segment displayed is increment by 1000h. If you display memory near or at the start of physical memory (0000:0000), the 0000 segment is automatically used. This area of memory contains the x86 interrupt vectors, which consist of 4-byte pointers to the programs that handle the interrupts.

Stack segment displays automatically display stack frames pointers in reverse video, and referenced locations in boldface (as in TRACE MODE).

In hex/ASCII mode, the Tab key switches back and forth between the hex and ASCII display columns.

Ctrl-A switches back and forth between the hex/ASCII and pure ASCII formats (used also for X window in TRACE MODE).

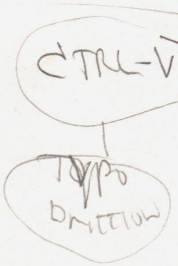
Ctrl-B switches to the Byte (hex/ASCII) format

Ctrl-E Ends the block at the location pointed to by the cursor. The other end of the block is defined by Ctrl-T.

Ctrl-G Goes back and forth between the Tag and End tag locations.

Ctrl-S inverts (Shifts) the case of the character at the cursor.

Ctrl-T Tags the cursor location for use by the Ctrl-G command and for defining one end of a block that can be used in **c**ompare, **d**isplay, **e**xamine, **f**ill, **m**ove, **s**earch, and **w**rite commands (see Block section of Chap. 3).



Ctrl-Z displays last char read in by the last **l**oad command executed

As for all commands, ↵ (or Esc) returns to COMMAND MODE (ASSEMBLE MODE typically takes two ↵'s)

Memory Pointers

SST uses certain control characters to specify that the bytes starting at the cursor are to be used as pointers into memory for subsequent display and disassembly. This helps one to move around in memory without typing addresses.

Ctrl-C Continuously updates the display. This is optional, since some screens glitch with this process, and the keyboard response may be slowed down. This feature is handy to watch areas of memory being changed by interrupt-driven routines, such as the time-of-day clock, and the keyboard input buffer. Try using it while displaying 40:0.

Ctrl-D displays memory at the Double-word address indicated by the cursor.

Ctrl-F starts disassembling memory at the Far address indicated by the cursor. This is very handy for looking at the code of an interrupt handler. Display memory at 0000:0000 (just type the **d**isplay command dd;, which acts like dd0000:0000), move the cursor to the desired interrupt vector and typed Ctrl-F. This displays the

first instruction of the interrupt handler. Typing the space bar or PgUp displays subsequent instructions.

Ctrl-N starts disassembling memory at the Near address indicated by the cursor.

Ctrl-O toggles between DISPLAY MODE and OVERTYPE MODE as described below.

Ctrl-U Undoes the last overtype in case you type something by mistake.

Ctrl-W displays memory in the current segment starting at the offset given by the Word-address indicated by the cursor.

Overtyping Memory with SST

By typing Ctrl-O, you toggle between DISPLAY MODE and OVERTYPE MODE. In the latter when the cursor is located in the hex columns, typing hex digits overtypes those in memory. When the cursor is on ASCII columns, typing characters with blank or larger ASCII codes overtypes memory. Control characters must be entered as hex values. Since you may overtype memory by mistake and not know what value you overtyped, the Ctrl-U option allows you to replace the value of the last location overtyped. Use the overtype facility with caution. You may overtype something you don't mean to.

SST display syntax is upward compatible with Microsoft's SYMDEB, except that the x87 modes remain with the examine command, and SYMDEB lacks the full screen mode. The command syntax sometimes gives different results from DEBUG.COM, but the result is always clear and the extra power is worth the change.

Binary Editor

If a non-com, non-exe file smaller than 64K bytes is read in, an elementary binary mode edit capability exists in addition to the usual overtype capability. The Del key deletes the byte at the cursor from the RAM image of the file, decrementing the user cx value accordingly. The Ins key inserts a binary null at the cursor, incrementing the user **cx** value accordingly. Such a file can be overtyped and bytes can be inserted and deleted, regardless of the file content, i.e., the file can have arbitrary binary values. The **w** command then rewrites the file with the new length back to the same place on disk (unless you use the **n** command to change the filename).

Displaying Labels

Labels can be displayed by a command of the form

▷d/*n*

where *n* is the number of a segment paragraph. Often the desired labels are in the current code segment, in which case type

▷d/cs

To **d**isplay program **v**ariable names, type

▷d/v

To **d**isplay **u**ser strings, type

▷d/u

To find out the address (segment:offset) of a given label, type *&label_name*.

DATE Command

The date command in COMMAND MODE displays the current system date.

DEL Command

A command of the form

▷del *filename*

deletes the file *filename*. This form could also have the unlikey interpretation as a Display command, starting at the offset 0Eh with the number of bytes specified after the "L". This possibility is superseded by the del command.

DELAY Command

A command of the form

▷delay *n*

causes the code

```
mov     cx,n
```

```
loop    $
```

to be executed each time most characters are displayed on the screen. This slows down SST displays, mostly to help in the debugging of SST itself.

DELETE Command

A command of the form

▷delete *n*

deletes the instruction at the offset *n*. This works only in Program mode (see Chap. 6).

DIR Command

Typing dir or ls in COMMAND MODE acts very much like typing dir/W at the DOS command prompt, but also displays labels, and system, hidden, and directory files. You can follow the dir command with an arbitrary filename specification including path and asterisks. For example, the command

▷dir *.asm

or just

▷dir .asm

displays all files in the default directory on the default drive with the extension .ASM. The filenames displayed are alphabetized. At the end of the display the total byte count for all file whose filenames are displayed is given in decimal if that number is less than 65536 or if the computer has an x87. Otherwise the count is given in hex.

DISK Command

The command disk changes the display source for the d command from RAM to disk. See Chap. 7 for details. The command ram switches the display source back to RAM.

DOS Command

The dos *n* command is the equivalent to typing v21 *n*00 at the COMMAND MODE prompt. The only advantage is that if you type it incorrectly you won't execute some undefined interrupt vector by a mistake.

DOUBLE Command

To facilitate both source-level and assembly language debugging, SST includes commands to define typical data types. The syntax for the double type is:

▷double [[far] *] *variable name address*

For example,

▷double alpha 305

adds the symbol alpha of type double (x87 64-bit floating-point) to the segment specified by the **ds** segment register at the offset 305h. You can specify any other segment. The optional * generates a near ptr to a variable of the type double. The optional far generates a far pointer to a variable of the type double.

DR Command

On computers based on 386 and later processors, the dr command displays the 80386 debug registers 0, 1, 2, 3, 6, and 7, the translate-lookaside registers 6 and 7, the control registers 0, 1, and 3, and the extended flags register. This command does not work when SST is run as a V86 task. It does work if SST is run in real or protected mode.

DWORD Command

To facilitate both source-level and assembly language debugging, SST includes commands to define typical data types. The syntax for the dword type is:

▷dword [[far] *] *variable name address*

For example,

▷dword alpha 305

adds the symbol alpha of type dword (32-bit unsigned integer) to the segment specified by the **ds** segment register at the offset 305h. You can specify any other

segment. The optional * generates a near ptr to a variable of the type dword. The optional far generates a far pointer to a variable of the type dword.

E Command

This command examines or modifies memory on a byte by byte basis (like DEBUG), in various floating point formats, and according to user defined templates. The display command can also be used change memory, but the examine command is occasionally preferable, since the screen remains largely unmodified and x87 data types are supported. To execute the command, type e or E plus an address and then hit the space bar. The system will respond by displaying the contents of memory at that address. Syntax:

▷e *address*↵

For example, typing e2000↵ results in

▷e2000 00-

if the contents of location 2000h are 00. Typing in a value *nn* at this point will change the contents of location 2000h to *nn*h, while hitting the space bar will leave the contents of that location alone and display the contents of the next higher location. One can continue entering new values or hitting the space bar as long as desired. The command is terminated by hitting ↵. For example,

▷e2000 00- 11-10 22- 33- 44-1210 55- 66-

would change locations 2001h and 2004h to 10 and leave the other locations unchanged. Note that the keyboard input routine uses only the last two characters typed before the space bar is hit. Thus in the above example, 12 was entered by mistake in location 2004h and then corrected by immediately typing 10 before the space bar was hit. One can also correct a mistake in the previous byte by pressing the backspace key to re-display the preceeding byte.

Floating Point Values

When an x87 is installed, the SST examine command can also be used to examine and change long integer, packed BCD, and floating-point values in memory. If e [*address*] is followed by a "/" and one of letters b, d, l, o, p, q, s, t, or w, memory is examined in the following formats respectively:

format	# bytes
BCD	10
Double precision real	8
Long integer	4
O binary	1
P binary	2
Quad integer	8
Single precision float	4
Temp precision	10
Word integer	2

For example, typing the command e100/d followed by a ↵, you might see

▷e100/d - 0

At this point if you type a number in like 1.2345, the 8 bytes at location 100h would be changed to the floating point number 1.2345. Subsequent typing of the space bar examines subsequent 8-byte double precision floating point quantities.

Structure Templates

Structure templates are used to display memory in customized formats that reveal the data in its natural form, rather than in one of the usual uniform formats like hex/ASCII. Such layouts include linked lists and data structures with mixed data types. The templates used to describe these data structures are mixtures of 1) alphanumeric names that begin with a letter, 2) single decimal digits, 3) $*n*, where *n* is a value < 100, 4) $b or $w, 5) >*n*, and 6) string literals '...'. The string names, contents of the string literals, and all other bytes are displayed as is. The digits and $ fields have special meanings as follows:

1, 5-9	display the next 1, 5-9 bytes in hex
2	display the next 2 bytes as a 16-bit word
3	display the next 3 bytes as a 24-bit word
4	display the next 4 bytes as a double word (segment:offset)
$*n*	display the next *n* ASCII characters from memory
$b	display the character string following the next byte in memory with length given by that byte
$w	display the character string following the next word in memory with length given by that word
$z	display null-terminated character string

$$	display $-terminated character string
>*n*	go forward the next *n* bytes
>b	go forward the number of bytes specified by the next byte in memory
>w	go forward the number of bytes specified by the next word in memory
>z	skip null-terminated character string
>$	skip $-terminated character string
<*n*	go backward the next *n* bytes
<b	go backward the number of bytes specified by the next byte in memory
<w	go backward the number of bytes specified by the next word in memory
<z	go backward to preceeding null-terminated character string
<$	go backward to preceeding $-terminated character string
=*n*	go to offset *n* in current segment
=w	go to offset specified by the next word in memory
=d	go to address specified by the next double word in memory
=s	go to offset 0 in segment specified by the next word in memory
/b	x87 BCD format (10 bytes)
/d	x87 Double precision (8 bytes)
/l	x87 Long integer (4 bytes)
/q	x87 Quad integer (8 bytes)
/s	x87 Single precision (4 bytes)
/t	x87 Temporary real (10 bytes)

The / options require the x87.

For example, to read out a descriptor data structure with the macroassembler form

```
dscptr  struc
sglen   dw      ?       ;Segment max length
sgbase  dw      ?       ;Segment base low word
        db      ?       ;Segment base high byte
access  db      ?       ;Segment access byte
reswrd  dw      ?       ;Reserved word
dscptr  ends
```

define the string ds by

```
▷nds="sglen    2
sgbase 3
access 1
reswrd 2"
```

Then use the examine command as follows

▷e *address*/ds

After the first ↵, subsequent space bars display the next structure entry in memory. User strings along with program labels, comments, and variables are all saved together by the wl, and can be reread by the ll command.

Useful DOS Examine Templates

A set of useful examine templates is given on the SST distribution diskette in the file STRUCT. These templates include those for the EXE header, the Program Segment Prefix (PSP), the File Control Block (FCB), the Extended FCB, the Drive Parameter Table (DPT), the Device Header, and the Bios Parameter Block. We are indebted to Guy Gordon of White Crane Systems for donating these templates for SST users.

The examine command is upward compatible with DEBUG.COM, except for the use of the backspace and extra digits in arguments. The command adds the ability to examine and change floating-point values in IEEE format and by user-defined structure templates.

ECHO Command

The echo of display and unassemble output to the printer or echo file set up by the n>*filename* command can be controlled by the echo command. Type

▷echo on

to turn it on and

▷echo off

to turn it off.

EDIT Command

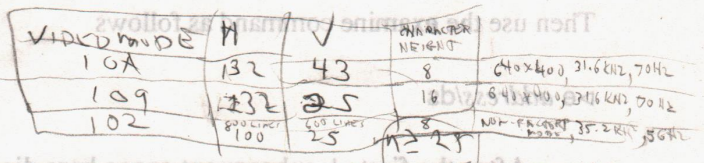
A command of the form

▷edit *address*

enters the ASSEMBLE EDIT MODE for the instruction at the address *address*. See the assemble command and Chap. 8 for further discussion.

EGA *n* Command

A command of the form



▷ega *n*

determines the Enhanced Graphics Adapter's line/page mode. *n* = 43 chooses the 43-line mode and *n* ≤ 25 chooses the 25-line mode.

ERASE Command

SST includes a subset of the MS-DOS file commands for speed and convenience (see Chap. 4). One of these is the erase command, which is typed in COMMAND MODE in the form

▷erase *filename*

After getting this command, SST asks you to confirm that you really want to erase the file *filename*. If you type y or Y, SST erases the file; otherwise it does not. You can interrogate the directory with the dir command in COMMAND MODE.

The alternate DOS form for erase, del, can be used in SST. This form could also have the unlikey interpretation as a Display command, starting at the offset 0Eh with the number of bytes specified after the "L". This possibility is superseded by the del command.

F Command

This command fills a block of memory with a constant of one or more bytes (like DEBUG). Syntax:

▷f *range list*

For example,

▷f100,1C0,FF

fills memory locations **ds**:100h through **ds**:1C0h with the hex value FF.

The command can also fill a block of memory with a list of assembly language instructions, handy for checking speed of execution. For this option, the list is replaced by @ ↵, which transfers control to the assembler. Type in the instructions you want followed by two ↵'s in a row. This fills the memory range you give repeatedly with the instructions you give. For example,

```
▷f 100 1FE @
1234:0100          loop 100
1234:0102
```

fills 100h through 1FEh with a loop to here instruction. This sort of fill command is useful for finding out how fast pieces of code run (be sure to turn off interrupts—see the qi *n* command).

The command is upward compatible with DEBUG.COM, and adds the ability to fill memory with a set of instructions.

FLOAT Command

To facilitate both source-level and assembly language debugging, SST includes commands to define typical data types. The syntax for the float type is:

▷float [[far] *] *variable name address*

For example,

```
▷float alpha 305
```

adds the symbol alpha of type float (x87 32-bit floating point) to the segment specified by the **ds** segment register at the offset 305h. You can specify any other segment. The optional * generates a near ptr to a variable of the type float. The optional far generates a far pointer to a variable of the type float.

G Command

The g command allows a user to go execute a program with breakpoints (like DEBUG). These breakpoints go away when SST regains control. Sticky breakpoints are also available as described under the breakpoint command in this chapter. In addition, the SST go and sticky breakpoints can be made conditional, that is, whether execution is stopped when the instruction at a breakpoint is reached can be made to depend on a set of conditions specified by the user. When

the **g**o command is executed, SST loads the values in the register storage area into the proper registers, and then jumps to the requested program address.

The go breakpoint syntax is:

▷g [=*address*] [*address*]...

For example,

▷g1000

starts execution at **cs**:**ip** and breaks if **cs**:1000 is reached.

▷g=1000,1020,es:1230

starts executing at **cs**:1000h, and breaks at **cs**:1020h or **es**:1230h, the program will return to the monitor, printing the register values. All register contents at the time of the breakpoint are saved. All previously set go breakpoints are cancelled when any breakpoint is reached. Breakpoints must be set only at locations corresponding to the first byte of an instruction. Additional breakpoint facilities are built into the screen trace mode, and greatly reduce the frequency that you need to use the unconditional **g**o command. The breakpoint facilities use the **int 3** instruction, and only work in RAM.

Conditional Breakpoints

To make the breakpoints depend on a set of conditions, follow the **g**o command specification (i.e., just before the ↵) by "@". This transfers control to the assembler to allow the set of conditions to be entered. The conditions are expressed by an arbitrary set of assembly language instructions. These instructions could in principle invoke software interrupts, call user subroutines, and do anything else that the machine can do. The conditions are specified the same way for the Super-Trace and for conditional breakpoints. Hence after a conditional breakpoint succeeds, a subsequent trace will automatically be a Super-Trace (note the S at the end of the second line from the top of the screen), unless the conditions are turned off with either a g@ ↵ ↵ or a t@ ↵ ↵ command.

There are three basic guidelines to writing conditional breakpoint code:

1. The **ax** and **bp** registers are saved before the user code is executed. It is your responsibility to save and restore any other registers you wish to use. The **bp** register is initialized to point to the program stack, with **bp-2** giving the user **ax** value, **bp+0** giving the **bp** value when the breakpoint was encountered, **bp+2** giving the **ip**, **bp+4** giving the **cs**, and **bp+6** giving the flags. The **ax**

register is initialized to the first word of the current user instruction. For example, if the current instruction is a **ret**, then **al** = 0C3h, something you can break on.

2. The area reserved for user code is 40h bytes long. This is more than enough if you plan to type conditions in hand, but could be easily exceeded if you load a .COM file into the condition memory (to find the address of this memory, type g@ ↵ ↵ or t@ ↵ ↵, which in addition to displaying the condition memory address turn off any active conditions). If you want to access a large amount of condition code such as a program profiler, make it a resident routine and access it through an **int** instruction.

3. Program execution is interrupted if the instructions set the Zero flag to 1, that is, if a **jz** instruction would jump. Otherwise program execution continues.

Writing Conditional Code

Chapter 3's section on "Super-trace Demonstration" illustrates the simple condition

```
cmp     di,600
```

which succeeds if and only if **di**=600. Sometimes it is easy to express a condition that yields NZ rather than Z. For example, suppose you want the condition that **di**≠600. For this use the code

```
cmp     di,600
lahf
test    ah,40
```

Here the lahf, test ah,40 complements the Zero flag. Since SST saves **ax** for you, you don't have to worry about clobbering the **ah** register. More complicated conditions often require some conditional jump instructions as well.

If you want to stop supertracing on the next **ret** instruction, use the condition

```
cmp     al,0C3
```

H Command

This command is included for primarily for compatibility with DEBUG.COM. The SST calculator provides a much more powerful facility. The **h**ex command adds and subtracts two hexadecimal numbers. Syntax:

▷h *value*$_1$ *value*$_2$

If the *value*$_2$ is missing, the binary equivalent of *value*$_1$ is displayed. To get hex/decimal conversions, see Sec..3-4 on the "Calculator".

Examples:

▷h345,abc E01 F889

▷h10= 00010000

This command is upward compatible with DEBUG.COM's, adding the hex to binary conversion facility (used mostly for tutorial purposes).

HELP Command

The help command subjects the file SST.HLP to the SST type command. This allows you to browse/search through the SST.HLP file looking for online help.

I Command

This command displays the binary value read from any **i**nput port (like DEBUG). Type i or l followed by the port number. Syntax:

▷i *portaddress*

Thus

▷i20

inputs a byte from input port 20h. The input value is dis⊟played in binary, i.e., if the value obtained from the input port was 45h, the display would show

▷i20 45=01000101

INI Command

The ini command reads and executes the file called SST.INI. This can be used to initialize SST for your standard set of parameters (see also the q command).

NOTE: If an sst.ini file exists in the default directory, it is automatically executed before SST displays its signon message.

INSERT Command

The command insert *n* enters the ASSEMBLE INSERT MODE at offset *n*. This works only in Program mode. See the assemble command and Chap. 6 for further details.

INT Command

To facilitate both source-level and assembly language debugging, SST includes commands to define typical data types. The syntax for the int type is:

▷int [[far] *] *variable name address*

For example,

▷int alpha 305

adds the symbol alpha of type int (16-bit signed integer) to the segment specified by the **ds** segment register at the offset 305h. You can specify any other segment. The optional * generates a near ptr to a variable of the type int. The optional far generates a far pointer to a variable of the type int.

INT21 Command

DOS **int 21h** entry definitions are displayed when you type the int21 [*n*] command in COMMAND MODE. If the optional *n* is present, the definition for that entry point alone is displayed. If *n* is missing the next hexadecade of **int 21** entries is displayed. In addition unassembled **int 21h** instructions are commented

with the corresponding entry descriptions. These features are very handy for working with code that makes DOS calls.

J Command

No commands currently begin with J.

K Command

The k command is used to clear (klear) the screen, to give a program stack trace, to reset the x87 registers and status, and to get keyboard input codes.

K - Stack Frame Display

A special stack readout may enable you to trace subroutine calls. Most higher level languages support a recursive subroutine linkage convention that creates a stack frame for each call. Unless the compiler is optimizing code, each subroutine call saves the current value of **bp** on the stack and points **bp** at the saved value. Use of **bp** then allows access to the subroutine arguments which have been pushed onto the stack before the call and to local variable storage on the stack which is allocated upon entry to the subroutine. Immediately above the save **bp** value is the Near or Far return address.

To display such a stack trace, type

```
▷k
```

in COMMAND MODE. Note that the k command can take arguments, which cause it to do other things as described below. For SST to display the trace, the numbers it encounters on the stack must make sense as stack frames. The **bp** register must contain a value at least as large as the **sp** register. The saved **bp** values must be greater than the current **bp** value and must increase monotonically. As soon as one of these requirements is violated, the trace terminates.

SST stack traces use the Microsoft Windows convention to determine if a return address is Near (16-bits) or Far (32-bits). Specifically, if the saved **bp** value is even (as it is whenever loaded as a frame pointer into **bp**), the return address on the stack is assumed to be a Near address, that is, 16-bits long. In contrast for a Far (32-bit) return address, the Microsoft Windows subroutine initialization code saves the **bp** value + 1, i.e., an odd value. SST's stack trace therefore assumes that an odd saved **bp** value signals the presence of a Far return address. The instruction preceeding that at the return address given in this fashion is examined to

see if it is an appropriate **call** instruction. If so, a call trace display is given and the next frame is examined. SST also displays up to eight stack values in between the frames.

For a more general call trace, special coordination between SST and the .EXE symbol tables is needed.

Klearing the screen and x87

The k command clears the screen (except for the register window at the top). This is useful when starting to assemble code following displays or traces that are irrelevant to your assembly. The cls command is an alias. This command does not exist in DEBUG.COM.

To **k**lear screen lines *n* through *m* (*n*=0 is screen top), type a command of the form

▷**k** *n,m*

To **k**lear the **f**loating point (x87) registers, type

▷kf

To display the **k**eyboard **i**nput code *c*, type a command of the form

▷ki ↵ *c*

where ↵ stands for the Enter key.

KEY Command

The key command modifies the control characters used to edit command lines. See Sec. 4-3 on *Modifying Edit Command Characters.*

KEYBOARD command

To deal with hostile keyboard environments, SST has a built-in **int 9** keyboard encoder. When debugging programs under Microsoft Windows or in Protected Virtual Address Mode, this keyboard facility is ordinarily on. Otherwise is is left off, and SST gets its keyboard input from **int 16**, unless keyboard redirection of some sort is enabled. A command of the form

▷keyboard *status*

enables or disables the built-in keyboard support if *status* = on or off, respectively. The facility doesn't handle the enhanced keyboard new keys.

KILL Command

The kill command works like the erase command to erase disk files, but allows the filename to be quoted (as in BASIC).

L Command

This command loads a file or absolute sectors (like DEBUG)

▷L [*address* [*drive sector count*]]

If the drive and sector specifications are missing, it loads file named by name command (see below) at the address specified on the l command line. If the address is missing, the file is loaded at cs:100. If the file has an .EXE extension, it is loaded as an .EXE file with appropriate address relocation and segment register initialization.

For example to name and load a program called TEST.COM, type

▷ntest.com
▷l

You can then type t or T to trace program execution, u or U to unassemble some code, or d or D to display the program in hex/ASCII.

If you make changes in the program, you can write the revised version back to disk using the write command. Be sure the bx and cx registers have the values they had when you loaded the file, since they determine how many bytes will be written. With SST you can not only read .EXE files as can DEBUG, but also write them back to disk. This is very handy for patching your favorite system programs.

The following error messages can occur:

File not found
Error in .EXE file

Load Labels

The load command is used also to load in program labels, variable names, and user macro and examine template strings. These facilities are described in greater detail in Secs. 4-4 and 4-5 on "Labels" and "User Strings and Keyboard Macros".

To load in program labels, name the .MAP file with the n command, and type the ll command (see load command). This automatically reads the labels in starting at the point in the .MAP file identified by the words "by Value" and relocates them relative to the origin of the .EXE module (program prefix segment paragraph + 10).

To load .MAP labels relative to some other paragraph, type LL *n*, where *n* is the desired paragraph number. This option is useful for debugging resident programs. To load .MAP files for use with .COM files, type lm, which automatically adds 100h to the label offsets.

SST has limited support for program variables with the lv option. This option loads the variables defined by the part of a MASM.EXE listing for a single segment. The program scans for the word "segment" and sets up program variable names up to the corresponding **ends** pseudo op.

The ll, lm, and lv options use the user program area to load in the .MAP and .LST files and hence overwrite whatever program might have been loaded in. Hence to debug a program, load in the label files first, and then the program.

LIST Command

The command

▷list [*address*]

unassembles the code starting at the address *address* if specified, at 100h if a .COM file, or at the initial **cs:ip** if an .EXE file. The command goes into the unassemble full screen mode automatically.

LLIST Command

The command

▷llist [*address*]

acts as the list command and echos the output to the lister.

LOAD Command

The load command is used to load in an SST Program saved by the save command. See Chap. 6 for details.

LONG Command

To facilitate both source-level and assembly language debugging, SST includes commands to define typical data types. The syntax for the long type is:

▷long [[far] *] *variable name address*

For example,

▷long alpha 305

adds the symbol alpha of type long (32-bit signed integer) to the segment specified by the **ds** segment register at the offset 305h. You can specify any other segment. The optional * generates a near ptr to a variable of the type long. The optional far generates a far pointer to a variable of the type long.

M Command

This command **m**oves a block of memory from one location to another (like DEBUG). The command expects original start address, original end address, and destination start address as arguments. Syntax:

▷m *range address*

For example,

▷m2200,2280,1000

moves the contents of memory contained in the block **ds**:2200h through **ds**:2280h (inclusive) to **ds**:1000h through **ds**:1080h. If the original and destination memory blocks do not overlap, the original memory block is left undisturbed. However the two memory blocks can overlap with no ill effects. For example,

▷m2200,2275,2203

moves the contents of **ds**:2200h through **ds**:2275h up by three bytes in memory to **ds**:2203h through **ds**:2278h. This command moves a maximum of 64K in any one move. As elsewhere, segment values can be used to override the default value in **ds**:

▷mcs:2200,2275,es:2203

moves **cs**:2200 through **cs**:2275 to the block starting at **es**:2203.

MAP Command

MS-DOS has a linked list distributed throughout the low 640K bytes of RAM that defines how the RAM is allocated. This list consists of 5-byte paragraph-aligned entries. These entries immediately preceed the memory block (also paragraph aligned) that they describe. The first byte is an "M" for all control blocks except for the last, which has a "Z" for the first byte. MZ are the initials for one of the principal architects (Mark Zbikowsky) of MS-DOS, and also appear as the first two initials in an .EXE file. The second and third bytes give the 16-bit paragraph that owns the block, and the fourth and fifth bytes give the length of the block in paragraphs.

Specifically, the command

▷map

displays the five bytes of all DOS memory-control-blocks each followed by a two-line display of the start of the memory block they describe.

MOUSE Command

The command

▷mouse *status*

turns mouse control on if *status* = on and off if *status* = off. The mouse is used to control cursor motion for the DISPLAY and TRACE MODEs.

N Command

This command **n**ames a file for reading or writing using MS-DOS **int 21** routines (like DEBUG). Syntax:

▷n *filespec*

names the file given by the file specification *filespec*. For example,

▷nmyfile.exe

sets up the load command to be able to read in the file MYFILE.EXE.

The name command stores the name at 81h in the program prefix as does DEBUG.COM and sets up the first and second filenames at the 5Ch and 6Ch File Control Block areas in the program prefix. Note: when you load a program and then go to it, it may use its own name unless you issue a second name command to rewrite the one used to load the program. For example, if you load PS's editor module, PS.COM (a version of PMATE.COM with menu macros), and then go to it, it will open a file called PS.COM, which looks bizarre to say the least, since it's a binary file. To avoid this, type n ↵, which gives no parameters, or type n followed by the desired parameters. n= displays the current command line.

Saving Display and Unassemble Output to File

To define a file for output from the display and unassemble commands, type

▷n> *filespec*

This sets up the file given by *filespec* to receive display and unassemble output when those commands are typed with a range, e.g., d20,30 or u100 l 30. If the filename already exists, you are asked the question, "Overwrite existing file (Y/N)?" In this case, the file is opened for display and unassemble output if and only if you type y or Y. The n> output can be toggled on and off by typing

▷n>

i.e., without a filename. If n> echo output is enabled, a E shows up at the end of the second line from the top of the screen. If you attempt to toggle the file echo on without having defined a file for output, you'll see the message

Echo file undefined

Defining User Strings

To define a user string, type a command of the form

▷n*st*="*string*"

This defines (**n**ames) the string *st* (two letters long) to have the value *string*. User strings are used for examine memory templates (see **e** command) and keyboard macros as described in Chap. 4.

This command is upward compatible with DEBUG.COM's and adds file echo capabilities.

NEW Command

The new command restores the registers to the values used upon running SST, deletes all labels, and enables demostration and program facilities. Essentially new returns SST to its initial state.

NMI Command

The command

▷nmi *status*

enables (*status* = on) or disables (*status* = off) NonMaskable Interrupts on return from SST to the user program.

NOT Command

The command

▷not *range*

not's all bits in the bytes in the range *range*.

O Command

This command allows you to output any value to an output port. Type o or O followed by the port number and the desired value to output in hex. Syntax:

▷o *portaddress list*

For example,

▷o20,7F

outputs the value 7Fh to output port 20h. The list can have many bytes given by combinations of hex and quoted values. Hex values larger than 0FF are treated as word values.

Output to ports 3F8 and 2F8 wait for the TRHE bit (bit 5 of port 3FD and 2FD, respectively) to go high, indicating that the serial port is ready to transmit.

This command is upward compatible with DEBUG.COM's, and includes the ability to output words at a time and to handshake on the IBM PC serial ports.

OPCODE Command

The command

▷opcode *n*

displays the (or a) mnemonic for the byte opcode given by *n*.

OR Command

The command

▷or *range list*

or's the bytes in the range *range* with the bytes in the list *list*. The *list* is repeated as often as necessary to cover the complete range. This command is similar to the fill command, but **or**'s the list into memory rather than overwriting the bytes in memory.

P Command

The **p** command allows you to **p**rotect memory from being referenced in continous and quiet tracing. It has the syntax

▷p *address*$_1$ *address*$_2$

where both addresses can have segment specifications. This allows up to the full megabyte of memory to be protected. If address$_2$ is missing, only address$_1$ is protected. In the continuous and quiet trace modes, if protected memory will be referenced by executing the next instruction, the trace halts and the message "Protected Memory Referenced, Continue Trace (Y/N)?" appears. If you type y or Y, the trace continues; else COMMAND MODE takes over. To turn memory protection off, type the p command with no addresses.

If you can type appropriate commands for Super-Trace, you can catch an undesired memory reference much faster than with this mode. If you can get near the bad reference with breakpoints, the continuous traces may give you just what you want.

PAUSE Command

The pause *n* command pauses a time proportional to *n* whenever a Ctrl-\ is encounted in the keyboard input. This is used to pace demonstrations such as that invoked by the A option on Function Key 7.

PROMPT Command

The prompt command changes the usual COMMAND MODE prompt to a MS-DOS-like pathname prompt. For example, if you're in subdirectory \BIN on drive C, the COMMAND MODE prompt becomes

```
C:\BIN>
```

This can be a bit confusing if that's the same prompt you use for DOS itself, but in any event you see the register window at the top of the screen, indicating that SST is active.

Q Command

This command is used to return to DOS and also to define a number of SST system parameters controlling screen attributes and other machine characteristics. To return to DOS, type

```
▷q
```

which asks if your want to leave SST. If you type y or Y, you get back to the DOS system command prompt, just like DEBUG.COM. You can also return to DOS by typing any of the commands quit, bye, or system. These commands do not require confirmation.

To return resident, i.e., with SST available by typing Ctrl-↵ or by pressing an NMI button, type

▷q/R

If you want the user screen to be saved in this mode, be sure to use the qs3 command below before typing the q/R.

Screen Characteristics

Various screen characteristics are defined by the following q commands:

▷q *c n*	Set screen attribute for window c = a, h, n, r, s, x, z, for Assemble, Help, Normal, Register, Stack, Xam, Zam
▷ql *n*	Set lines/page = *n*
▷qy *n*	Set # lines Xam window = *n*

To see an example of configuring the colors attributes for various SST windows, see the section "Configuring SST" in Chap. 3

The qn *n* command is used to set the **n**ormal screen attribute and also to control screen "snow" for the ancient IBM color/graphics type displays. SST automatically recognizes the IBM color/graphics display and eliminates most of the snow. qn80 *xx* sets the **n**ormal screen attribute to *xx* and suppresses deglitching (good, e.g., for COMPAQ, which doesn't have **snow**). This speeds up screen displays by about a factor of three. qn40 *xx* forces deglitching.

The qs *n* command with *n* missing or less than 4 is used for various screen save and switching options described below. To set the stack window attribute to a number less than four, set the high bit of the word to 1. For example to set the stack attribute to green on a black background, type

▷qs8002

Screen Save

SST's screen saving facility is described in Sec. "Screen Save Option" in Chap. 3. The command

```
▷qs3
```

turns on the screen save option if enough room is allocated (see sst/*n* option in the "Command Line Parameters" section of Chap. 3 to increase this allocation). The command

```
▷qs2
```

turns off the screen save feature. The COMMAND MODE or TRACE MODE V option switches to the saved user screen. Typing any character thereafter returns to the mode before the V option was chosen.

Where to Display SST

SST display output can be sent to many different places to allow maximum flexibility in debugging programs that themselves use the screen. The options allow you to display SST output in various parts of a given screen (particularly useful on screens larger than 25 lines), on different screens, and in arbitrary parts of memory for use with nonstandard video RAM and multitasking window programs.

To set the origin of the SST display output for a given screen to line *n*, type

```
▷qo n
```

To display SST output in the lower half of 66 line screen, type

```
▷qol
```

To swap IBM monchrome and EGA/VGA displays for SST alone, type

```
▷qs
```

To swap IBM monchrome and EGA/EGA displays for both SST and DOS, type

```
▷qs1
```

SST choses the screen RAM segment by consulting **int 10h** on the IBM PC machines. To overrule this choice, you can set the segment to the paragraph *n* by a command of the form

▷qg *n*

Similarly the 6845 CRT controller port is chosen according to **int 10h** information, but can be overruled by a command of the form

▷qp *n*

which sets 6845 CRT I/O port = *n*. The IBM PC monochrome display has the value 3B4h and the color/graphics display has the value 3D4h, both of which are recognized automatically by SST. To configure on some other machine, you may need this command.

Interrupt Mask

The command

▷qi *mask*

sets the interrupt controller mask to the value *mask*. This is useful in debugging multitasking systems in which the system clock might be used to switch tasks after SST gains control. For example, to allow only keyboard interrupts use qiFD.

QUIT Command

The command

▷quit

returns to MS-DOS. bye, system, and Alt-X do the same thing.

R Command

The r command allows you to examine and change the contents of the x86 registers and flags as for DEBUG.COM. Under SST the r command is basically useless, since the register and flag value are always displayed in the register window at the top of the screen, and the values can be changed by more easily by simple assignments like

ax=100

The usual DEBUG command

▷r

displays the current register values in the COMMAND window. This is useful for echoing the results of a debug session to a file or printer.

Changing Register Values

As for DEBUG,

▷r *register*

displays the contents of the register *register*. Entering a new value *nn* from the keyboard enters a new value for the register. For example, entering rdx when **dx**=0000 results in the display

▷rdx 0000-

If *nnnn* is now typed, **dx** will have the new value *nnnn*h. If you do not want to modify the value, hit the space bar to display the next register or hit return to terminate the command.

Alternatively, the command

▷*register* = *value*

sets the register *register* equal to the value *value*. Valid register names are **ax**, **bx**, **cx**, **dx**, **al**, **bl**, **cl**, **dl**, **ah**, **bh**, **ch**, **dh**, **si**, **di**, **bp**, **sp**, **eax**, **ebx**, **ecx**, **edx**, **esi**, **edi**, **ebp**, **esp**, **ds**, **es**, **cs**, **ss**, **fl**, and **ip**.

With SST the x87 floating point stack values can be changed by typing

▷s *n* = *value*

where s *n* can be s0 through s7.

The flags Auxiliary Carry, Carry, Parity, Sign, Zero, Direction, Interrupt Enable, Overflow, and Trap can be set to 1 or reset to 0 by typing their leading letter followed by f as in

▷cf=1

which sets the **c**arry **f**lag to 1.

The register command is upward compatible with DEBUG.COM's. Its display differs in that the most recent values alone are always displayed at the top of the screen. This approach is much easier on the eyes than DEBUG.COM's, which constantly scrolls the screen. Note that in TRACE MODE, SST can retrace up to 20 steps (or more—see Sec. 3-2 on "Command Line Parameters"), allowing you to see earlier values of the registers.

Real Mode

SST can run in the 8086 Real Address Mode or on x86 machines with x > 1 in Protected Virtual Address Mode (see Y286 command in this chapter). To return to real mode after running in Virtual Mode, type the command

▷rm

This restores SST segments to those appropriate for a .COM file.

Restoring Registers and NMI Interrupt

To restore registers to their values when the last .COM or .EXE file was loaded, type the command

▷rr

This is handy for rerunning a program after examining how it terminated.

To return the NMI interrupt vector back to the program used before SST was loaded, type the command

▷rn

We find this handy for debugging SST itself, and it is also useful for tricky situations when a special hardware debugger has advantages over a software debugger.

The command r *n* lets you toggle SST's windows on and off. See the subsection on "Trace-Mode Hot Keys" for hot keys 0-8 under the T command below.

RAM Command

The RAM command returns the display source to RAM from disk as established by the disk command. See Chap. 9 for details.

REDIT Command

The redit command puts the cursor into the register window, where you can overtype register values. The facility is also available in TRACE MODE by typing the "e" hot key.

REN Command

The command

▷ren *file$_1$ file$_2$*

renames the file *file$_1$* to *file$_2$* like the corresponding DOS command.

RUN Command

The run command restores the registers to their initial values and transfers control to the program entry point.

S Command

This command searches for a string of characters or bytes (like DEBUG), or for a string of assembly language instructions. Syntax:

▷s *range list*

where *list* can be composed of one or more strings of the form "..." and bytes consisting of one or two hexadecimal digits. For example with **ds**=1234,

▷s100 4000 1A 3E "abc"

would display

1234:856 A0C FFE 3254

if the string of five bytes 1A 3E 61 62 63 starts at the locations 1234:856, 1234:A0C, etc. If the string of bytes is not found, SSt displays the message

String not found

For convenience, two abbreviated forms of the search command are included. Typing s or S with a range only searches that range for the last string en-

tered. This allows you to change the range of your search. Typing s or S alone repeats the last search. This is handy after the original search hits are scrolled off the screen. The search command quits with Ctrl-C and pauses with Ctrl-S.

Searching for Assembly Language

To search for assembly language instructions, type @ ↵ for the list field. This leads to the same kind of display and entry as given by the assemble command. The list of instructions is terminated by a hitting ↵ twice as for the assemble command, and is followed by a display of all addresses (if any) where the instructions given are found. For example,

```
▷s cs:0 1000 @
1111:0088       mov     ax,1
1111:008B       push    ax
```

searches the memory from **cs**:0 to **cs**:1000 for the instructions that push a 1 onto the stack (with the SST running on an 80186 or later processor, you can type push 1 for this, but that might not correspond to the code you're searching). Here the 1111:0088 is a sample starting location of SST's search string memory. SST's data segment value 1111: will almost certainly be something different when you run SST.

Being able to search for assembly language mnemonics is very useful for debugging programs consisting of many separately assembled modules, since you typically only know the addresses of code relative to the module origins.

Searching for Jumps/Calls to Location

SST also allows you to find all jump and call references to a particular program offset within a range of memory. Type

▷s *range* j *n*

This lists the offsets (relative to the segment register given by *range*) of instructions that jump to the offset *n*. The instructions checked for are: near/far direct **call**, short/near/far direct **jmp**, the 17 conditional jumps like **jz**, and the three loops. Indirect jumps and calls are not checked.

This option differs from the corresponding DEBUG option in that twelve addresses are displayed per line instead of one and in its ability to search for assembly language instructions.

SAVE Command

The save command is used to save an SST Program as a COM file with a special header allowing labels and comments to be saved as well. Subsequently, such COM files can be loaded in by the load command. See Chap. 6 for details.

SNOW Command

The ancient Color/Graphics Adapter snow can be controlled by the command

▷snow *status*

which turns CGA snow check on (*status* = on) or off (*status* = off).

SYSTEM Command

The command

▷system

returns to DOS. bye, quit, and Alt-X do the same thing.

T Command

This command traces program execution with full screen displays (unlike DEBUG). Syntax:

▷t [*n*]

turns on the TRACE MODE. If *n* is missing, the trace starts at **cs**:**ip**; if *n* is present, the trace starts at *n*. TRACE MODE displays the **r**egisters as for the r command followed by the menu

TRACE MODE: **F1 B**reak **# G**o **C**ont **D!F**ast **S**low **J**mp **N**op **R**e **W**in **T!X**am x87

and a screenful of machine language and disassembled instructions starting at the starting trace address. The current instruction line is highlighted by a reverse video

bar. Each depression of the space bar single steps the program. Typing Function Key 1 displays the help screen

Break	Break at IP	**# n**	break at IP after n passes
Go adr	break at adr	**Line**	break at next source Line
Here	break at cursor	**Point**	toggle breakPoint at cursor
Fast	break following IP	**Slow**	trace IP
Space	Single step	**Don't**	call single step
Undo	last instruction	**Nop**	skip IP
Jmp	Jump unconditionally	**Kick**	IP to cursor
Quiet	trace	**Cont**	Continuous trace
@scii	display ASCII screen	**+-&**	source/asm/mix
Ice	stack	**Offset**	change stack readout offset
Overtyp	stack/xam window	**Asm**	at cursor
Edit	registers	**Z**	x87 status
ASCII	toggle ASCII vs HEX	**Txam**	toggle Tracking
F5	zoom window	**F6**	change window
↑↓	scroll window	**PgUp/Dn**	scroll window
W/X	toggle Program/Xam	**View**	program window
2/7/8	toggle menu/x87/stack	**0/1/3**	no/8086/80386
Redraw	screen	**Enter**	→ COMMAND MODE

TRACE MODE Hotkeys

In alphabetical (ASCII) order, the hot keys are defined as follows:

Ctrl-A toggles ASCII vs hex/ASCII display modes in the memory eXamine window.

Ctrl-B forces hex/ASCII (Byte) display mode in the memory eXamine window.

Ctrl-D moves the top bar of the program-output **d**own if the cursor is in the program-output window. If the cursor is any other window, the memory eXamine window's top bar is moved **d**own.

Enter returns to COMMAND MODE

Ctrl-O zooms the cursor's window directly into OVERTYPE MODE. Typing the Esc key, the Enter (↵) key, or Function Key 5 again returns to TRACE MODE.

Ctrl-U moves the top bar of the program-output **up** if the cursor is in the program-output window. If the cursor is any other window, the memory eXamine window's top bar is moved **up**.

Space single steps the program, and ↵ returns to COMMAND MODE.

Note: on all x86 microprocessors manufactured after 1981, single step doesn't stop until two instructions after a segment modification instruction like mov ds,ax. On the 80386, single-step may step two iterations of a rep string instruction.

displays

iterations =

to which you type the number of times, *n*, the current instruction should be allowed to execute before breaking (equivalent to typing *n* B's for break).

&-+ switch between assemble/source modes. **-** switches to pure assembly mnemonics, **+** switches to pure source code, and **&** displays mixed source and assembly mnemonics.

0-8, w, x toggle the display of SST windows as follows:

Key	Effect
0	toggle register window
1	set 16-bit register display
2	toggle menu window
3	32-bit register display
7	toggle x87 window
8	toggle program stack window
w	toggle program output window
x	toggle Xamine window

If option **7** is used with no x87, the error message "No x87 installed" is displayed. The x87 condition codes and register stack are displayed below the program stack. See the **z**amine command for a complete description. Since ordinary decimal and scientific notation is used, this display makes debugging x87 code fairly easy (with DEBUG.COM it's essentially impossible).

@SCII displays the ASCII help screens (see Chap. 2 for full description).

Assemble (a or A) switches to ASSEMBLE EDIT MODE at address at the cursor. The Enter key enters the current instruction and goes onto the next. The Esc key returns to TRACE MODE without entering the current line.

Break executes the instruction at **cs:ip** and then sets a breakpoint the execution on the next encounter (RAM only).

Continuous runs continuously until a key is depressed. The continous trace stops if protected memory (see **p**rotect command) is referenced, or if an illegal op code is encountered, or if an instruction for a higher-level machine is attempted, e.g., running a **pusha** (push all) on an 8088.

Don't single-step calls executes call instructions at full speed by setting a breakpoint following the call instruction. On other instructions it simply single steps like the space bar. This differs from the Fast hot key, which executes any current instruction at full speed by putting a breakpoint after that instruction. The **d**on't hot key allows you to see the general flow of a routine without getting sidetracked down subroutines. The Don't option marks the first 10 subroutines it encounters as Don't-trace subroutines. These subroutines can be returned to trace mode by using the Slow option.

Edit (e or E) switches into the register window and allows you to overtype register values, and the Zero, Carry, and Sign flags. Use the arrow and tab keys to move around the window. Type the Enter key to enter the new values and continue tracing. Type the Esc or Ctrl-C keys to suppress the new values and continue tracing.

Fast executes the current instruction at full machine speed, breaking when encountering the instruction after the current instruction (RAM only). This is useful for calling a subroutine or finishing a loop instruction without single-stepping through it. Note that if the current instruction is a jump, the fast hot key may amount to a go.

Go *address* sets a breakpoint at the address *address* while remaining in TRACE MODE. This saves the effort of returning to COMMAND MODE and then to TRACE MODE when you know the breakpoint address you need. In addition, G*s, G*b, G*c set temporary breakpoints at [**sp**], [**bp**], and offset at stack-window cursor, respectively and go. The stack values used are near addresses. To specify

far addresses, use G*fs, G*fb, and G*cb, respectively. G*s is useful for breakpointing upon returning from a subroutine. Be sure that the return address is in fact at [**sp**]. Another way to return from a subroutine if you haven't executed too far into it is to **U**ndo back out of the subroutine and **D**on't call around it.

Here sets a temporary breakpoint at trace-window cursor position and goes. This hot key is available in UNASSEMBLE MODE as well.

Ice *ices* the stack readout offsets at their current values. This iced mode is indicated by reverse video offsets.

Jmp causes an unconditional jump (useful for overruling a conditional jump) to **cs:ip+(ip+1)**.

Kick *kicks* the Instruction Pointer to the address at the cursor. Only the instruction pointer is changed by this hot key. This hot key is available in UNASSEMBLE MODE as well.

Line single-steps with no screen updated until the next source-code line is encountered.

Nop skips the next instruction altogether (but doesn't change the code).

Offset cycles between stack offset value modes. These offsets can be made relative to **ss**:0, to **sp**, or to **bp**. Stack segment displays show all stack frames in range in reverse video and a referenced location in bold. This feature also works in DISPLAY MODE when the segment displayed is the same as that given by the stack segment register **ss**. SST also bolds the target offset of a conditional jump that will jump.

Point toggles the sticky breakpoint at the cursor position. This hot key is available in UNASSEMBLE MODE as well.

Quiet toggles quiet continuous trace. This mode only updates the registers and runs about three times as fast as the **C**ontinuous trace. The quiet trace stops if protected memory (see **p**rotect command) is referenced.

Re Redraws the screen with the currrent instruction at the top. This is handy if the current instruction is displayed at or near the bottom of the display and you want to see the following instructions.

Slow single-steps the next instruction, which lets you trace execution of an **int** instruction (normally executed in **F**ast mode).

Txam toggles the Tracking feature of the eXamine window. When tracking is on a T appears at the end of the second line from the screen top. The eXamine window always displays the memory around the last location referenced by the program, and the cursor identifies this location. This is a useful feature and leads to fascinating demos when run continuously.

Undo Undoes the last single step. This can be repeated up to 20 times (or more - see Chap. 3), literally allowing you to see your program execute backwards. This is very useful for recalling the steps that lead to an anomalous condition.

View switches to the user screen if the screen save option is enabled (see the qs3 command)

Win toggles a Window 15 lines down from the screen top for MS-DOS CRT output. This is handy for debugging routines that write a moderate amount of text to the screen using standard system calls.

Xam toggles a two-line memory eXamine window at the bottom of the screen. This window displays 20h bytes in hex/ASCII format and 80h bytes in pure ASCII format. If the window is not in tracking mode (see next option), you can scroll through memory using the arrow keys and the PgUp PgDn keys. The cursor is displayed at the last location referenced in the window.

Z gives you a pop-up full screen of information about the x87 status. Typing x or X when this Z screen is present shows you the heX values of the x87 floating-point registers, regardless of whether they are tagged "empty", or invalid.

Trace Mode Window Control

Function Key 5 zooms the stack/xamine windows into DISPLAY MODE. This gives a full screen with full DISPLAY MODE features including overtype capability. The Ctrl-O hot key zooms these windows directly into OVERTYPE

MODE. Typing the Esc key, the Enter key, or Function Key 5 again returns to TRACE MODE.

Funtion Key 6 switches between windows in TRACE MODE. The window with the cursor can be scrolled up and down with arrow, PgUp, and PgDn keys. When in the trace window, the up and down-arrow keys are useful in combination with the **A**ssemble, **H**ere, **P**oint, and **K**ick hot keys. To toggle windows on and off, see the **0-8** hot keys.

Super-Trace

Section "Super-Trace Demonstration" of Chap. 3 describes a special SST facility called Super-Trace. This facility single steps a program in a very tight loop, executing a set of user-specified conditions after each single step. These conditions are written in ordinary assembly language and are assembled by the **a**ssemble command module. The requirements for the code are given under the Conditional Breakpoint section of the **g**o command. Basically **ax** and **bp** are saved before entering the user code, and **bp** points at the program stack and **ax** has the first word of the current instruction. No return instruction is necessary, since SST automatically supplies the return. If the code sets the Zero flag, SST takes over, allowing the user to examine the machine. If the Zero flag is reset to 0, the Super-Trace continues. Typically Super-Trace runs at about one tenth full machine speed (in real mode, not v86 mode), although this depends markedly on how much code the user specifies for conditions. If a whole execution profile routine is called, execution could easily be slowed down another factor of ten.

TIME Command

The time command displays the current time of day as calculated by the computer.

TRACE Command

The trace command restores the registers to their initial values and goes into TRACE MODE at the program start address.

TYPE Command

The SST type command displays a file in a full-screen menu-driven mode that scrolls forward and backward with the PgUp, PgDn, and up and down arrow keys,

goes to the start/end of the file with the Home/End key, toggles the display of line numbers with the # key, goes to line *n* with the **L**ine option, and searches forward or backward for an arbitrary literal string.

To display the file *filename* in this mode, type a command of the form

▷type *filename*

Typing the command

▷type

with no argument displays the file previously typed at the same location that you left it. To leave the TYPE MODE, press the Esc key.

U Command

This command **u**nassembles machine-language instructions. Syntax:

▷u *address*

unassembles the instruction at the address *address* and goes into UNASSEMBLE MODE. If *address* is missing, the instruction following the last one unassembled is unassembled, or if no previous **u**nassemble command has been executed the instruction at **cs**:**ip** is used. Typing the space bar unassembles the next instruction. Typing a PgDn unassembles a whole screenful (except for the register window at the top of the screen). PgUp does a rudimentary Page Up procedure that subtracts a number of bytes from the current unassemble address and unassembles a screeful from that lower address. To be sure you're properly synchronized, type a few space bars. ↵ and Esc go back to COMMAND MODE. The unassemble display format is the same as that for the **a**ssemble instruction illustrated above.

Alternatively

▷u *range*

unassembles the instructions within the range specified. This version of the **u**nassemble command works as for DEBUG.COM, while the other two options are different. The output of this start/end **u**nassemble option can be written to a file of your choice (see n> filename command).

USE16/32 Commands

The commands use16 and use32 control the 80386 segment D bit of the assembler. The default is use16, which generates code according to the standard 8086 segment addressing. The use32 command switches to the 80386 32-bit addressing mode, which allows access to 4-gigabyte segments, greatly increased indexing facilities, etc. See the Intel *Programmer's Reference Manual* for a detailed discussion of these modes.

V Command

The v command alone (followed by a ↵) swaps to the user screen if enabled by the qs3 option. Alternatively the **V** option in TRACE MODE flips between SST and user screens (on the same monitor).

When followed by a hexadecimal number, the **v** command calls an interrupt vector. Syntax:

▷v *n* [ax [bx [cx [dx]]]]

where *n* is the desired interrupt vector number, and the indicated registers are assigned values optionally. Current register values are used for values not given on the command line.

For example,

▷v21,600,,,7

rings the bell, since an **ah**=6 **int 21** instruction outputs the character in **dl** (here 7) to the system console. After returning from the vector command, the user **ax**, **bx**, **cx**, **dx**, and flags are updated to show what called interrupt vector did.

The vector command saves and restores the user screen if the save screen option is enabled by typing qs3 in COMMAND MODE.

Virtual Mode

On the PCs with 80286 and later microprocessors, you can enter the Protected Virtual Address Mode by the command

▷vm

and return to Real Address Mode by the command

▷rm

The Protected Virtual Address Mode gives you direct access to the ISA-bus's entire 16 megabyte address space as well as to various protected mode features. On EISA bus systems, you can access the full 4-GB address space. In paged mode operation (the usual case under Microsoft Windows), you view a linear address space, rather than the physical address space, so even with ISA-bus systems, it appears as if you can see a full 4-GB address space (see Chap. 5 of Sargent and Shoemaker (1994) for a detailed explanation). You can use SST to debug .COM files in this mode, run programs in extended memory up above the Real Address Mode's megabyte, examine memory in extended memory, and so on. There are limitations as to what you can do with protected mode, especially if you run under a protected-mode operating system. In general, the only protected-mode system that you can run SST in protected mode is Microsoft Windows 3.x in an enhanced-mode DOS box.

If you run SST in "real" real mode, that is, not in V86 mode as happens if emm386.sys is installed or if you executed in a Windows DOS box, then your program can change the Interrupt Descriptor Table origin from what SST sets up, but must leave the IDT descriptor in the GDT at offset 90h. In this real real-mode operation, the SST vm command sets up user stack selector = 60h, data selectors = 68h and code selector at 70h. This works for .COM and .EXE files.

The Ctrl-Enter key interrupts vm operation as in Real Mode and Ctrl-Alt-Del works. In addition sticky breakpoints work. In protected modes, the DOS dir, chdir, prompt, and type commands work, although not in a completely general way. After typing vm in COMMAND MODE, which switches to Protected Virtual Address Mode, typing any one of these DOS-like commands invisibly switches back to Real Mode, calls needed DOS commands, displays the desired information, and then switches back to protected mode. The screen pretends that protected mode is always enabled (V at lower right of the register window).

In some kinds of protected mode operation, SST's built-in keyboard (**int 9**) program is used. The time-of-day clock is also turned on to convince you that the machine is still running.

The SST protected mode tracks Real/Virtual Mode on trace/breakpoint options. Hence if you leave SST in Real Mode, and SST traps an interrupt occurs in protected mode, SST automatically switches itself to protected mode. Similarly if you leave SST in protected and reenter in Real Mode, SST switches itself to Real Mode operation. Note that protected mode operation requires careful coordination between SST and the underlying operating system. The only ways SST currently works in protected mode is when it starts in real mode (not V86 mode) or when it is run in a Windows DOS box.

W Command

This command writes a file or absolute disk sectors (like DEBUG). Syntax:

▷w [*address* [*drive sector, count*]]

If the drive and sector specifications are missing, it writes the file named by the name command (see above) at the address specified on the l command line. If the address is missing, the file is written starting from cs:100.

.EXE files can also be written provided they are read in first. This allows you to patch an .EXE file. Be sure not to change the segment specification values inadvertently. The following error messages can occur:

No room in disk directory
Insufficient disk space
Insufficient memory
Error in .EXE file
Read .EXE file before writing

Write is very useful for modifying a disk file. Name the file with the name command, load it with the load command, make the changes you want being sure not to change the values of the bx and cx registers, and then type w or W to Write the modified version back to disk.

Write Labels

The labels, variable names, and user strings currently defined can be written to the file named by the last n command by typing

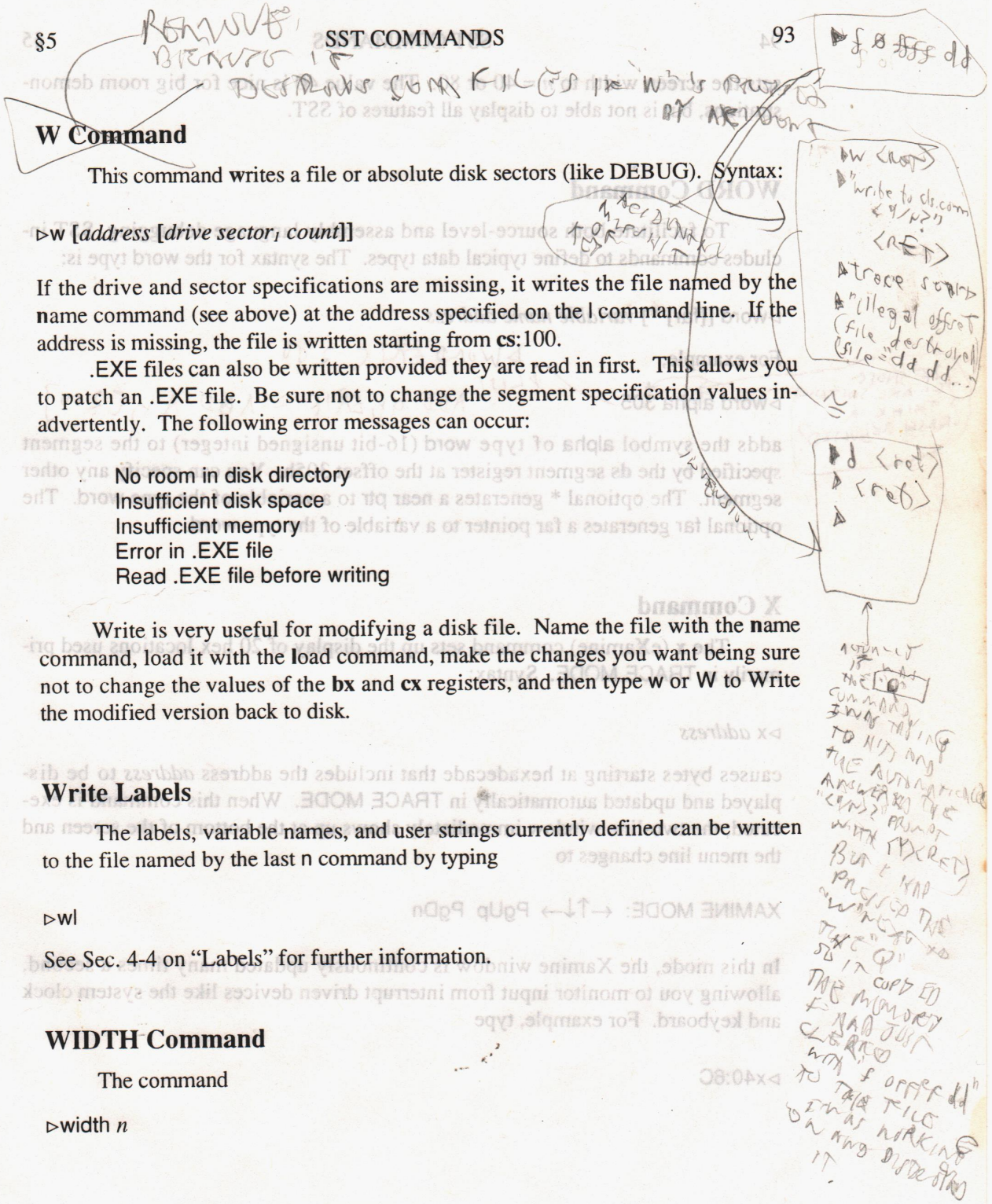
▷wl

See Sec. 4-4 on "Labels" for further information.

WIDTH Command

The command

▷width *n*



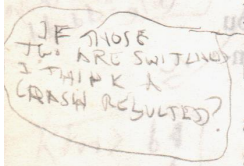
sets the screen width to *n* = 40 or 80. The value 40 is nice for big room demonstrations, but is not able to display all features of SST.

WORD Command

To facilitate both source-level and assembly language debugging, SST includes commands to define typical data types. The syntax for the word type is:

▷word [[far] *] *variable name address*

For example,



▷word alpha 305

adds the symbol alpha of type word (16-bit unsigned integer) to the segment specified by the **ds** segment register at the offset 305h. You can specify any other segment. The optional * generates a near ptr to a variable of the type word. The optional far generates a far pointer to a variable of the type word.

X Command

The **x** (eXamine) command sets up the display of 20 hex locations used primarily in TRACE MODE. Syntax:

▷x *address*

causes bytes starting at hexadecade that includes the address *address* to be displayed and updated automatically in TRACE MODE. When this command is executed, the two-line window immediately shows up at the bottom of the screen and the menu line changes to

XAMINE MODE: ←↑↓→ PgUp PgDn

In this mode, the Xamine window is continously updated many times a second, allowing you to monitor input from interrupt driven devices like the system clock and keyboard. For example, type

▷x40:6C

and watch the clock tick away. Scroll up in memory to see the keyboard input queue change as you type the right arrow. In either XAMINE MODE or TRACE MODE, the up and down arrows scroll the memory display up (towards smaller memory addresses) and down respectively. The PgUp and PgDn keys scroll up and down by 100 hex at a time.

XOR Command

The command

▷xor *range list*

xor's the bytes in the range *range* with the bytes in the list *list*. The *list* is repeated as often as necessary to cover the complete range. This command is similar to the **f**ill command, but **or**'s the list into memory rather than overwriting the bytes in memory.

Y Command

Development of protected-mode applications requires a careful understanding of how the Global and Local Descriptor Tables work. In SST Protected Modes, the segment values for **ds**, **cs**, **es**, and **ss**, along with many internal values have be changed to correspond to values in SST's vm Global Descriptor Table (GDT). Notice after executing the virtual **m**ode command that the segment register values are relatively small numbers. These correspond to entries in the GDT. To read the GDT table, use the y command as follows:

▷y *n*

If *n* is present, list the GDT (or LDT) entry *n*; if not, list GDT entries one per space bar. RAM and Global Descriptor Table displays beyond the segment limit have the offset field displayed in reverse video.

Defining Global Descriptor Table Descriptors

To define new descriptors to refer to areas of memory of your choice use the command (in general this is only available when SST starts execution in "real" real mode)

▷y *n address* [*access* [*length*]]

This defines a GDT entry 70<= *n* <=D8 at *address*, access=*access*, with length=*length*. If the *access* and *length* fields are missing, 93h is assumed for the access (writable data segment) and a segment length of 0FFFFh is used for *length*.

For example, to be able to examine, compare, move, etc. memory starting at the second megabyte in physical memory, type the command

▷y70 100000

Then in protected mode, the command

▷d70;

will display a full screen displaying this RAM. If you have a VDISK installed, you'll see the VDISK copyright notice.

Z Command

This SST command displays the complete x87 state in greater detail than in the TRACE MODE. If you attempt to use the x87 facility without an x87, you see the pop-up message

```
No x87 installed
Better go get one!
```

Trace Mode 7 Option

Typing 7 in TRACE MODE toggles the x87 window underneath the program stack window. The x87 window has the x87 condition codes on top followed by the values of the eight x87 80-bit registers. They are displayed with st(0) on top and with nine decimal places for integers and for typical floating point values, and about five for those requiring exponent notation. For example, you might see a window like

```
      1001
 12
 123456789
 1.25000000
-123.456000
-1.23456e-4
 empty
```

```
empty
empty
```

The 1001 tell you the x87 condition code bits **c3**, **c2**, **c1**, and **c0**, respectively, which reflect the results of x87 compare, test, examine and remainder instructions. Other status bits can follow as discussed under "x87 Status Bits" below. The word empty means that the corresponding stack registers have not been loaded. Some other special values such as infinity and unnormal are labeled accordingly.

Trace Mode Z Option

For more accuracy in either TRACE or COMMAND MODE's, type z or Z, which gives the full 80-bit values in scientific notation along with some help information. For the example above, you'd see the screen

```
The x87 is set with projective infinity, full precision, and round to even

0i

stack index = 0    cc=1001

st(0) = 12                          Status codes:
st(1) = 123456789                     P-Precision exception
st(2) = 1.2500000000000000            U-Underflow
st(3) =-1.2345600000000000            O-Overflow
st(4) =-1.2340000000000e-4            Z-Zero divide
st(5) = empty                         D-Denormalized operand
st(6) = empty                         I-Invalid x87 opcode
st(7) = empty

Letters above the bar indicate normal interrupts, below indicate masked
interrupts.  Type x or X for heX display of registers.

Type any key to continue
```

x87 Hexadecimal Display

Typing x or X replaces the help on the right by the hexadecimal values of the registers. The screen above changes to

```
The x87 is set with projective infinity, full precision, and round to even
i
stack index = 0    cc=1001
st(0) = 12                          0 4002  C000000000000000
st(1) = 123456789                   0 401D  932C05A400000000
st(2) = 1.2500000000000000          0 3FFF  A000000000000001
st(3) =-1.2345600000000000          1 4005  F6E978D4FDF3B647
st(4) =-1.2340000000000e-4          1 3FF2  8164EF6DE184EAB8
st(5) = empty                       1 795F  DD768A987E5689F2
st(6) = empty                       1 795F  DD768A987E5689F2
st(7) = empty                       1 795F  DD768A987E5689F2
Letters above the bar indicate normal interrupts, below indicate masked
interrupts.  Type x or X for heX display of registers.
Type any key to continue
```

Note that the empty registers do have values, although they don't mean anything. The values may be left over from earlier computations. Registers with invalid contents are flagged by "??" which have special hex values identifying the nature of the problem. These special values are easily examined with the zamine X option.

x87 Status Bits

The condition code bits, interrupt request bit, and exception flag bits from the x87 status word are reported immediately above the register stack. The binary values of the four condition code bits are always displayed at the upper left of the x87 window. These bits reflect the results of x87 compare, test, examine, and remainder instructions. The other bits are displayed by letter if they equal 1, and are represented by blanks if they equal 0. A pending interrupt request is displayed as an "i" following the condition codes. The six exception flags are identified by the corresponding capital letters in the following list: Precision, Underflow, Overflow, Zerodivide, Denormalized operand, Invalid operation. For example, you may see a P fairly often, since precision exceptions are not unusual. For a detailed discussion of these bits, please consult one of the Intel manuals on the x87 numeric coprocessors.

6. Assembly Language Interpreter

SST has a simple, built-in assembly-language interpreter. Typically this interpreter mimics the BASIC interpreter, except that it expects assembly language statements instead of BASIC statements. It also differs from previous interpreters in a number of ways, such as having the full power of a screen debugger and using native machine code as the intermediate interpreter language, which can lead to faster programs than those from compilers let alone usual interpreters.

The BASIC-like word commands coexist with the DEBUG-style single letter commands remarkably peacefully. Words like load, save, list, llist, run, and delete are syntactically illegal from DEBUG's point of view, and hence can be used unambiguously directly in SST's COMMAND MODE. The BASIC command new is ambiguous, since to debug it means name the file called ew, but if you really want to name such a file you could type n ew, which is not recognized as new. A complete list of such command appears under the heading "Interpreter Commands" in this chapter.

To see a demonstration of the interpreter among other things, use the SST Auto demo option given by typing Function Key 7.

6-1. Line Numbers

BASIC uses statement line numbers for branching and editing purposes. Similarly the assembly language interpreter instructions are automatically located in memory and can be referred to by their hexadecimal memory offset values. You use these offset values like line numbers to insert, delete, edit, trace, and execute instructions. Since many instructions are longer than one byte, there are many illegal "line numbers" referring to the middles of instructions. The assembly language interpreter tells you if you try to refer to one of these illegal numbers.

To make sure that it knows what's an instruction without undue overhead, the interpreter insists that its code area (code segment) contains only instructions. If you use a db or dw pseudo op to define variable storage, that storage will automatically be allocated to the program data segment, rather than to the code segment. The interpreter has a very fast algorithm for scanning through a program up to the end statement that allows it to check for legal line numbers. This same al-

gorithm is used to insert, delete and overtype instructions, all of which can involve shifting the code up or down in memory. When you make a change in a program, the interpreter *reassembles* the code at about 11,000 instructions a second on an ordinary PC. Actually it doesn't have to completely reassemble the code; it only has to shift the code as needed and update all relative offsets in **jmp** and **call** instructions appropriately.

6-2. Labels

The assembly language interpreter allows the use of labels for referring to variables and jump addresses. As you type in or list a program, references to undefined labels are stamped with a "U" to the left of the corresponding machine code. When you resolve these references by typing in a statement with a missing label, the references are filled in. For example, you can type call alpha, where alpha hasn't been defined previously, and then later type in the subroutine called alpha. If you subsequently delete with instruction with the label alpha, all corresponding references are stamped as Undefined, until you redefine their target again.

Instructions and Pseudo Ops

The interpreter accepts all x86 instructions as well as the three pseudo ops end, db, and dw. The pseudo op end is used to specify the end of the code. Typically you don't have to use the end pseudo op, since the interpreter knows where your code ends. However if you want to delete the code from some point through the end of what you've typed in, you can type end in sooner. To start over, you should use the word new instead, since that also deletes the labels you've typed in.

The db and dw pseudo ops are used to define program variables and assign them initial values. For example,

```
message db "This is a message",0
```

defines memory for the user variable message. The trace command reinitializes all variables to the values given by the db and dw pseudo ops.

6-3. Edit Command

The SST edit facility can be used to edit assembly language instructions. To edit a line, type

▷edit *line_number*

in COMMAND MODE. This switches to ASSEMBLE MODE and automatically calls up the line with the *line_number* (instruction offset) specified. Make the changes you want and type ↵ to go onto the next line. To quit editing, type Esc, which returns to COMMAND MODE.

While in UNASSEMBLE and TRACE MODEs, you can also move the cursor in the trace window to any instruction and switch to ASSEMBLE EDIT MODE by typing the hot key "a".

6-4. Interpreter Commands

SST recognizes the following DOS/BASIC-like commands while in COMMAND MODE (see also Chap. 5).

bye	return to DOS
close	close all disk files
cls	clear screen
cont	continue execution at full speed (not tracing)
delete *n*	delete instruction at offset *n*
edit *n*	edit statement at offset *n*
files *template*	list directory with template *template*
insert *n*	insert instruction(s) starting at offset *n*
list [*n*]	list (display) program from start [from offset *n*]
llist [*n*]	print program from start [from offset *n*]
load *file*	load file with filename *file*.COM
new	delete all labels, restore initial registers values
run	run program from start at full speed (not trace)
save *file*	save file with filename *file*.COM
system	return to DOS
trace [*n*]	trace program from start [from offset *n*]

These commands exist in similar forms in DOS or BASIC. SST accepts relaxed syntax. For example, on the files command, you can enclose the filename template in double quotes or not as you choose. The insert command is added since legal line numbers always correspond to addresses of current instructions. On the other hand, renumbering is automatic, so BASIC's renumber command is superfluous. In addition to these commands, you have, of course, the standard SST commands, which can also be useful, particularly the a (assemble) and t (trace) commands.

7. Disk Display/Modify Facility

SST has a disk display/modify facility that works essentially like the memory display/modify facility except that you specify sectors instead of segments. The facility uses a 64K RAM buffer directly following the user program segment prefix, thereby overwriting anything you may have read in there. You can display the sectors in any of the standard SST display formats using the d command and scroll through the entire disk if you have enough time.

To switch into DISK DISPLAY MODE, type

▷disk [*sector:offset*]

where the *sector:offset* specification is optional. Leaving it out starts displaying at sector 0, offset 0. To return to memory display mode, type RAM in COMMAND MODE.

Overtyping Disk

You can switch into DISK OVERTYPE MODE by typing Ctrl-O and overwrite the disk image in memory. To update the corresponding sector on disk, type Ctrl-X, which asks if you want to overwrite the sector in question. Type y or Y to confirm the overwrite request and the disk sector will be overwritten.

Pointer Facilities

Some pointer facilities are available to help you move rapidly from one part of the disk to another by using the hierarchical directory structure. These are defined as follows:

Ctrl-P	Display root directory in current display format.
Ctrl-D	Display cluster corresponding to Directory Entry (DE) at cursor or if in FAT to cluster identified by cursor. Saves current disk location so that Ctrl-G returns to this location.
Ctrl-C	Display cluster chain for file described by DE at cursor.

Ctrl-L List DE in human-oriented form in pop-up window. While this window is active, typing down (up) arrow moves to the next (previous) DE. When pointing at FAT, Link cluster into cluster chain for file chosen by Ctrl-C option.

We recommend displaying the root directory and the subdirectories in pure ASCII format and use the Ctrl-L option to obtain more specific information. When you want to examine a file or subdirectory, position the cursor somewhere on the corresponding Directory Entry and type Ctrl-D.

File Allocation Table (FAT)

The disk space is assigned to files by use of linked chains of clusters stored in the FAT (File Allocation Table). A cluster consists of one or more sectors, 2 on a 360K floppy, 4 on the 20M AT hard disk. The FAT itself starts at sector 1. If you display the FAT in word format on hard disks with more than 10 megabytes or in triple-nibble (**dp**) format for floppies and smaller hard disks, the disk cluster chain pointed to by the cursor is highlighted. This is both instructive and useful, since you can see how the disk space is allocated. The appropriate display formats are automatically used when the Ctrl-F FAT display command is typed in the DISK DISPLAY MODEs. The offset field is also treated specially to give the cluster value in reverse video, rather than the FAT sector offset.

You can change the cluster allocation if you want to unerase a file or to construct a file from data on the disk. Move the cursor to the directory entry for the desired file, switch into DISK OVERTYPE MODE by typing the Ctrl-O toggle, type Ctrl-C to display its cluster chain, position the cursor at the desired cluster position and type Ctrl-D. To update the disk FAT, type Ctrl-X as described in the Overtying Disk section. You can also modify the chains by overtyping in hex/ASCII mode, but this is hard to decipher, expecially for 12-bit FAT formats (diskettes and smaller hard disks).

The disk display/modify facilility doesn't qualify as a full disk utility package since unerasing a file is not yet menu driven. Nevertheless the display capabilities typically exceed other disk utility packages both in sheer speed and in the variety of display formats which include, for example, **u**nassemble and **a**ssemble facilities. For more discussion, see Sections 12-2 and 12-3 of *The Personal Computer from the Inside Out, 3rd Ed.* (1994) by Sargent and Shoemaker.

Scroll Systems Inc

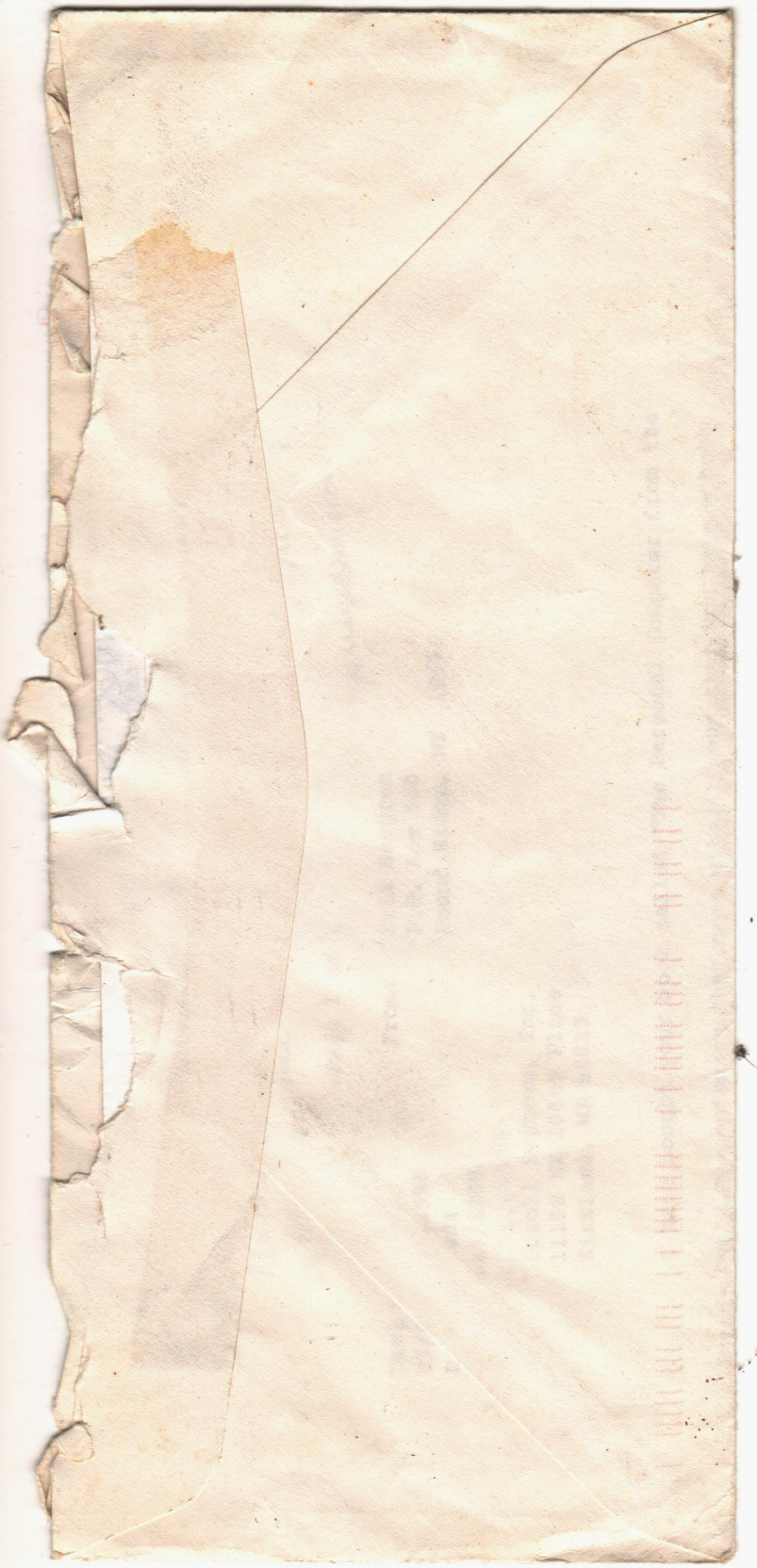
Mr. Murray Sargent, III
11108 NE 106th Pl.
Kirkland, WA 98033-5084

SEATTLE WA 981
PM
20 MAR
1998

USA 32

Erik Bovgren
P.O. Box 540
Pound Ridge, NY 10576

10576/0540



Erik,
No changes have been made. The manual is still available at $25. Thanks for your interest!
Murray
murraya@microsoft.com

Date written:
03/07(sat)/1998
Date mailed:
03/07(sat)/1998

From: Erik Bergren
P.O. Box 540
Pound Ridge, NY 10576

RECEIVED 03/31/1998 +10 day

To: Scroll Systems, Inc,
11108 NE 106th Place
Kirkland, WA 98033-5084

I have bought your book called "The Personal Computer from the Inside Out". The software that came with it says that you have a book called "SST Manual", and that it is available from the address above, for a price of $25.00.

Since that software file was written on 10/09/94, I want to check to make sure the book is still available, and what is the price you currently want for it. Also: if the program "sst" is available in a newer version, with less buggs, please tell me the price.

Thankyou,
Sincerely,

Erik Bergren

Date written:
03/07(sat)/1998

Date mailed:
03/07(sat)/1998

From: Erik Bergren
P.O. Box 540
Pound Ridge, NY 10576

To: Scroll Systems, Inc.
11108 NE 106th Place
Kirkland, WA 98033

I have bought your book called "The Personal Computer from the Inside Out". The software that came with it says that you have a book called "BST Manual", and that it is available from the address above, for a price of $25.00.

Since that software file was written on 10/09/94, I want to check to make sure the book is still available, and what is the price you currently want for it. Also: if the program "ast" is available in a newer version, with less bugs, please tell me the price.

Thankyou,
Sincerely,

Erik Bergren