Periscope

The Undercover Debugger
For The IBM Personal Computer

by Brett Salter

PERISCOPE
Version 1.0

Published by
Data Base Decisions
14 Bonnie Lane
Atlanta, GA 30328
U.S.A.
(404) 256-3860

The hardware is warranted to be free from defects for one year from the date of purchase. If you need to return the hardware for repairs, please call the telephone number above for a return authorization number.

The software is provided **AS IS**, without warranty of any kind, either expressed or implied. We will attempt to fix any problems you encounter with the software if you will notify us of the problem and provide us a way to recreate the situation in which the problem occurs. We welcome your suggestions and comments regarding improvements and enhancements to the software.

The entire risk as to the performance of this package is with the purchaser. Data Base Decisions has carefully reviewed the materials provided with this package, but does not warrant that the operation of the hardware and software will be uninterrupted or error-free. Data Base Decisions assumes no responsibility or liability of any kind for errors in the package or for the consequences of any such errors.

Contents

Page

Chapter I
Introduction 1-1
Overview 1-1
Features & Benefits 1-2
System Requirements 1-6

Chapter II
Getting Started 2-1

Chapter III
SUBMARINE — The memory board 3-1
Checking For Conflicts 3-1
Setting The DIP Switches 3-2
Board Installation 3-5

Chapter IV
Tutorial 4-1

Chapter V
PERISCOPE — The Debugger 5-1
Installation Options 5-1
Record Table Size 5-6
Symbol Table Size 5-7
The Command List 5-8
Keyboard Usage 5-9
Debugger Parameters 5-11
The Debug Commands 5-14
- Help 5-14
- Display Breakpoints 5-15
- Clear Breakpoints 5-16
- Breakpoint on Byte 5-17
- Breakpoint on Code 5-18
- Breakpoint on Memory 5-19
- Breakpoint on Register 5-20
- Breakpoint on Word 5-21
- Compare 5-22
- Display using Byte format 5-23
- Display using Double word format 5-25
- Display using Record format 5-26

Display using Word format 5-28
Enter 5-29
Enter Symbol 5-30
Fill 5-31
Go 5-32
Go using Breakpoints 5-34
Go using Trace 5-35
Hex arithmetic 5-36
Input 5-37
Jump 5-38
Klear 5-39
Load Absolute disk sectors 5-40
Load File from disk 5-41
Move 5-42
Name 5-43
Output 5-44
Quit 5-45
Register 5-46
Search 5-49
Search for Address reference 5-50
Trace 5-51
Unassemble 5-52
Write Absolute disk sectors 5-54
Write File to disk 5-55
Xlate (translate) hex number 5-56
Xlate (translate) Address 5-56
Xlate (translate) Decimal number 5-57
Option S 5-58

Chapter VI
RUN — The Program Loader 6-1

Chapter VII
Technical Notes 7-1
Debugging Theory 7-1
Debugging Techniques 7-2
PERISCOPE's Internals 7-4
The Memory Board 7-6

Appendices
Error Messages A-1
Index X-1

I — Introduction

Overview

PERISCOPE is a full-featured symbolic debugger, system monitor and **break-out** switch for the IBM PC and XT. This package includes a special memory board named SUBMARINE that is installed in one of the computer's expansion slots, a remote push-button switch, the debugger software, and this manual.

The memory card contains 16K of **write-protected** memory. Write-protected means that the memory can be written or changed only when the appropriate value has been output to the write-protect port on the memory card. The resident portion of the debugger software is loaded into this protected memory.

To be compatible with any possible PC configuration, DIP switches are used to select the memory and I/O ports used by the board. SUBMARINE is shipped with the DIP switches set to use 16K of memory starting at segment C000H or absolute address C0000H. The board uses two consecutive I/O ports — the default port setting is 300H. If the memory setting or the I/O port setting are in conflict with your configuration, you'll need to reset the DIP switches. See Chapter III for instructions on changing the switch settings.

The debugger software is loaded into the protected memory using a program named PERISCOPE (PS.COM). This program contains two parts — the transient portion and the resident portion. The transient portion copies the resident portion from RAM to the protected memory and then reserves a minimum of 10K of RAM as workspace for screens and symbol and record definition tables.

After PERISCOPE has been loaded, the push-button switch located on the board's mounting bracket can be used to interrupt the program currently executing. This switch generates a non-maskable interrupt or NMI. This interrupt is the highest priority of any activity in the system and as such is able to stop the system immediately. Do not press the

switch before PERISCOPE is installed or a parity error will be generated. PERISCOPE detects whether the parity error was generated by the button or if it was a memory error and displays an appropriate message.

The remote switch plugs into the phono jack located on the board's mounting bracket. This switch performs exactly the same function as the switch located on the mounting bracket.

To debug a program, the program RUN (RUN.COM) is used. This program loads your COM or EXE file into memory, reads the symbol and record definition tables if available, and transfers control to the resident debugger. This has the same effect as if you loaded the program normally and pressed the button just as the program began execution — the only difference is that the symbol and record definition tables are not loaded.

Features & Benefits

The push-button switch allows you to immediately stop the computer and examine the state of the machine. For example, if you're executing a program or the system is hung up, you can push the button to stop the system. In most cases you can recover the machine without having to turn the power off and back on again — saving the contents of memory and the stress induced by power on.

When the button is pressed, the current screen is saved and PERISCOPE's command list is displayed. This menu allows you to choose to perform any of the following: boot the system (same as Alt-Ctrl-Del), continue, enter the resident debugger, return to DOS, or perform a 'short' boot (Interrupt 19H).

One of the two boot commands is used when the system is hopelessly confused and there is no way to fix it. For example, if DOS has been clobbered, one of the two boot commands will usually need to be used. The short boot is the same as the normal boot, except that memory beyond the first 48K is not cleared. This option can be used to preserve data when certain RAM disk software is used.

The continue command restores the original screen and continues executing the original program where it was interrupted. The return to DOS command returns immediately to DOS. This function has certain requirements. See Chapter V for more information.

The resident debugger allows you to perform many functions, including:

- Set 'sticky' and temporary code breakpoints
- Set breakpoints on register values, byte and word values, and reads/writes to a range of memory, using Boolean comparisons. For example, you can set a breakpoint when register CX equals 0FFFH, or when a word in memory is less than some value. These breakpoints cause the system to run much slower than normal, but when they're needed, they can be invaluable.
- Compare two blocks of memory
- Display memory in three formats (byte, word, and double word), skipping lines of repetitive values
- Display a block of memory by individual fields within a record
- Enter changes to memory
- Fill a block of memory with a string/byte pattern
- Perform HEX addition, subtraction, multiplication, and division
- Read and write I/O ports
- Jump to the next instruction at the same level — useful for skipping over subroutine calls and interrupts
- Clear the debugger screen
- Read and write disk files and absolute disk sectors
- Move a block of memory to another location in memory
- Quit the debugger
- Display and change the registers and flags
- Search memory for a string/byte pattern
- Search memory for procedure and address references

- Trace one or more instructions
- Disassemble any location in memory, using symbols if a symbol table is loaded
- Display the names of the record and symbol table entries
- Interactively add and change symbol table entries
- Display help for any of the above commands

The debugger is symbolic, i.e. it allows you to use names from your program for the above functions. For example, if you have a subroutine named **PRINT_LINE**, you can Go to the first invocation of the subroutine by typing **G @PRINT_LINE**. If you disassemble the program in memory, the start of PRINT_LINE is shown as well as any data elements that PRINT_LINE uses. This symbolic capability can speed up debugging tremendously, since you do not have to look for a particular sequence of instructions to find a certain section of code, nor do you have to worry about the location of code or data — you can access it by name! You can also define new symbols or redefine existing symbols while the debugger is active.

If your compiler generates line number references in the object files it produces, you can also debug your program by the source line number.

In addition to symbolic input, you can use current register values as input to the debugger. For example, to display memory in word format starting at DS:SI, enter **DW DS:SI**. Or, to display the current stack, enter **DW SS:SP**.

PERISCOPE saves your current screen on entry and restores it when complete. When the Trace or Go commands are used, the original screen is restored before the Trace or Go, the Trace or Go is performed, and then the debugger screen is restored. This allows you to efficiently debug screen-intensive programs. If you have one monitor, you can switch between screens with a single keystroke. If you have both a color and a monochrome monitor, you can run your program on one monitor while debugging it on the other.

When PERISCOPE is activated, it saves the state of the machine, including the keyboard buffer, CRT control table, and other information. Since

it uses BIOS function calls for all but the Load, Name, and Write commands, PERISCOPE can be used to trace DOS as well as BIOS. When you push the button to interrupt a program, chances are very good that you'll stop the machine in DOS or BIOS. If DOS calls were used in PERISCOPE, you'd quickly run into problems, due to the fact that DOS is not re-entrant.

PERISCOPE accepts multiple commands on one line and remembers the previous command. Using these features, you can easily trace through a program, while monitoring an area of memory. It also supports the most commonly-used DOS editing capabilities, giving you the capability to edit command lines and print individual lines or the full screen to a parallel printer.

For the current instruction, any memory reads and/or writes are shown — even the effect of stack or string primitive operations. PERISCOPE also evaluates conditional jumps and tells you whether or not the jump will be taken.

PERISCOPE can search memory for references to procedures or data variables, showing you the instructions that reference the desired item.

The record display command allows you to display any area of memory using a previously defined format. This feature lets you quickly make sense of complicated records and structures.

If a memory parity error occurs, you can examine the instruction that caused the error to determine the failing block of memory, and then use the continue option to retry the memory read as many times as you want.

One offbeat use for the push-button switch is to temporarily silence a noisy printer if you get a telephone call while printing a long document!

PERISCOPE does not support 8087 mnemonics for disassembly of memory.

System Requirements

The system requirements for this package are:

- IBM Personal Computer or PC XT
- Any version of PC-DOS
- 64K RAM
- 1 Disk drive
- 80-column monitor

II — Getting Started

The enclosed diskette contains the files shown below. When you receive this package, backup the diskette using the following procedure:

- Place your DOS diskette in drive A and enter **DISKCOPY A: B:**.
- When prompted, insert the PERISCOPE diskette in drive A, and a blank diskette in drive B. Press any key to perform the copy.
- When DISKCOPY is complete, remove the original diskette and store it in a safe place.

The files on the diskette are:

PS.COM — This program loads the resident debugger into the protected memory. It is usually run once per DOS session, but can be rerun to change the size of its external data areas. These external data areas are used for the original screen (4 to 32K), the debugger screen (4K), the record definition table (1 to 32K), the symbol table (1 to 63K), and the optional help file (actual size).

PSHELP.TXT — This is the optional help file that is loaded into RAM by PS.COM when the /H option is specified. This file is a normal ASCII text file which can be modified as needed using a text editor.

READ.ME — This file, if present, contains any changes made to to this manual since it was last published.

RS.COM — This program is used to determine the record table size required by a DEF file. It enables you to efficiently allocate space for the record definition table.

RUN.COM — This is the program loader. It is used to load a COM or EXE file into memory, read the program's symbol table and record definition table if available and pass control to the resident debugger (PS.COM). This program will not work unless the SUBMARINE board

is installed and PS.COM has been run.

SAMPLE.ASM — This is the source code for the sample assembly program used in the tutorial.

SAMPLE.COM — This is the executable code for the sample assembly program used in the tutorial.

SAMPLE.DEF — This ASCII text file contains some sample record definitions. To use it with your program, change the name **SAMPLE** to the name of your program, leaving the file extension set to **DEF**.

SAMPLE.MAP — This is the MAP file produced when the sample program is linked. It is used to provide symbolic references.

TS.COM — This program is used to determine the symbol table size required by a MAP file. It enables you to efficiently allocate space for the symbol table.

To use this package, you'll need to perform the following steps:

- Install the protected memory board (Chapter III)
- Take the tutorial (Chapter IV)
- Install the resident debugger software using PS.COM (Chapter V)
- Load the program to be debugged using RUN.COM (Chapter VI)

Each of these procedures is explained in the following chapters.

III — SUBMARINE — The memory board

This chapter describes the installation of the protected memory board, SUBMARINE. The following topics are covered:

- Checking for conflicts with memory and port use
- Setting the DIP switches
- Board installation

Checking For Conflicts

The board uses 16K of memory and two consecutive I/O ports. For proper operation, the memory and ports used by the board must not be used by any other expansion option. When you receive the board, the DIP (Dual In-line Package) switches are set to use memory from C000:0000 to C000:3FFF and I/O ports 300H and 301H.

This area of memory is just above the area reserved for screen buffers, but below the area used by the fixed disk controller in the XT. The use of this area of memory may conflict with certain add-on memory cards that use this area for a ram disk or ROM device drivers. Check the documentation for any non-IBM expansion options to see if this is the case.

The two I/O ports used by the board are in the block (300H to 31FH) reserved by IBM for a prototype card. If you have a prototype card in your system, you'll need to check to see which ports, if any, it uses. Also, other expansion options may use these I/O ports. Check the documentation for any non-IBM expansion options to see if this is the case. Note that the true range of I/O ports available is from zero to 3FFH, since the IBM PC only supports the ten low-order bits of a port address.

If you find no documented conflicts with the memory or I/O ports used by the board, you can skip the next section on setting the DIP switches and proceed to the section titled Board Installation.

Setting The DIP Switches

There are two DIP switches on the SUBMARINE board. The switch labeled SW1 (nearer to the top of the board) controls the I/O ports used by the board. The switch labeled SW2 controls the starting address of memory used by the board.

SW1 is preset to use I/O ports 300H and 301H. The switches may be set to indicate any two consecutive I/O ports on a 4 byte boundary. You must not set the switches so that they conflict with other ports in the system.

Certain ports are off-limits. These include zero to 200H and others used by expansion cards in your system. Consult the IBM Technical Reference Manual and the documentation for your non-IBM expansion cards.

The switches can be read by laying the board on a table, component side up, with the top of the board facing you and the mounting bracket to your left. From this vantage point, the address can be read as a binary number. Switches eight and seven make up the first hex number, switches six, five, four and three make up the second hex number, and switches two and one make up the two high bits of the third hex number. When the switch is OFF (up or away from you), it corresponds to a one. When the switch is ON (down or towards you), it corresponds to a zero.

The three hexadecimal numbers correspond to the port address, 00xx yyyy zz00, where 00xx is the first number, yyyy is the second number, and zz00 is the third number. For example, when you receive the board, switches seven and eight are OFF (equal to one) and all others are ON (equal to zero). This is the same as the bit pattern 0011 0000 0000, which is 300H.

To change the port setting to 304H, move switch one to the OFF position. Notice that the two missing bits of the first and third numbers are always zero.

The table below illustrates the switch settings. The first section of the table illustrates the use of switches seven and eight to set the 'hundreds' part of the address, the second section illustrates the use of switches three through six to set the 'tens' part of the address, and the third section illustrates the use of switches one and two to set the 'units' part of the address.

DIP SWITCH SW1 (I/O PORT)

Starting Port Number	'Hundreds' S8	S7	'Tens' S6	S5	S4	S3	'Units' S2	S1
000	ON	ON	ON	ON	ON	ON	ON	ON
100	ON	OFF	ON	ON	ON	ON	ON	ON
200	OFF	ON	ON	ON	ON	ON	ON	ON
300	OFF	OFF	ON	ON	ON	ON	ON	ON
300	OFF	OFF	ON	ON	ON	ON	ON	ON
310	OFF	OFF	ON	ON	ON	OFF	ON	ON
320	OFF	OFF	ON	ON	OFF	ON	ON	ON
330	OFF	OFF	ON	ON	OFF	OFF	ON	ON
340	OFF	OFF	ON	OFF	ON	ON	ON	ON
350	OFF	OFF	ON	OFF	ON	OFF	ON	ON
360	OFF	OFF	ON	OFF	OFF	ON	ON	ON
370	OFF	OFF	ON	OFF	OFF	OFF	ON	ON
380	OFF	OFF	OFF	ON	ON	ON	ON	ON
390	OFF	OFF	OFF	ON	ON	OFF	ON	ON
3A0	OFF	OFF	OFF	ON	OFF	ON	ON	ON
3B0	OFF	OFF	OFF	ON	OFF	OFF	ON	ON
3C0	OFF	OFF	OFF	OFF	ON	ON	ON	ON
3D0	OFF	OFF	OFF	OFF	ON	OFF	ON	ON
3E0	OFF	OFF	OFF	OFF	OFF	ON	ON	ON
3F0	OFF	OFF	OFF	OFF	OFF	OFF	ON	ON
300	OFF	OFF	ON	ON	ON	ON	ON	ON
304	OFF	OFF	ON	ON	ON	ON	ON	OFF
308	OFF	OFF	ON	ON	ON	ON	OFF	ON
30C	OFF	OFF	ON	ON	ON	ON	OFF	OFF

DIP switch SW2 is preset to use memory starting at C000:0000 for 16K. The switches may be set to indicate any area in memory on a 16K boundary. You must not set the switches so that they conflict with other memory installed in the system.

Certain ranges of memory are off-limits. These include — 0000:0000 to x000:0000, where x is the number of 64K banks of RAM installed; B000:0000 to B000:0FFF if a monochrome adaptor is installed; B800:0000 to B800:3FFF if a color/graphics adaptor is installed; C800:0000 to E000:FFFF if the system is an XT; and F000:0000 to F000:FFFF on all systems. For example, if your system has 512K, memory below 8000:0000 is already used by RAM, and is therefore off-limits.

The starting memory address can be read by laying the board on a table, component side up, with the top of the board facing you and the mounting bracket to your left. From this vantage point, the address can be read as a binary number. Switches six, five, four, and three make up the first hex number, and switches two and one make up the two high bits of the second hex number. When the switch is OFF (up or away from you), it corresponds to a one. When the switch is ON (down or towards you), it corresponds to a zero.

The two hexadecimal numbers correspond to the highest part of the absolute memory address, xy000 or xy00:0000, where x is the first number and y is the second number. For example, when you receive the board, switches five and six are OFF (equal to one) and all others are ON (equal to zero). This is the same as the bit pattern 1100 00, which corresponds to C000:0000.

To change the memory setting to C400:0000, move switch one to the OFF position. Notice that the two missing bits of the second number are always zero.

The table below illustrates the switch settings. The first section of the table illustrates the use of switches three through six to set the 64K boundary, and the second section of the table illustrates the use of switches one and two to set the 16K boundary.

DIP SWITCH SW2 (MEMORY)

Starting Address	64K Boundary S6	S5	S4	S3	16K Boundary S2	S1
0000:0000 (0K)	ON	ON	ON	ON	ON	ON
1000:0000 (64K)	ON	ON	ON	OFF	ON	ON
2000:0000 (128K)	ON	ON	OFF	ON	ON	ON
3000:0000 (192K)	ON	ON	OFF	OFF	ON	ON
4000:0000 (256K)	ON	OFF	ON	ON	ON	ON
5000:0000 (320K)	ON	OFF	ON	OFF	ON	ON
6000:0000 (384K)	ON	OFF	OFF	ON	ON	ON
7000:0000 (448K)	ON	OFF	OFF	OFF	ON	ON
8000:0000 (512K)	OFF	ON	ON	ON	ON	ON
9000:0000 (576K)	OFF	ON	ON	OFF	ON	ON
A000:0000 (640K)	OFF	ON	OFF	ON	ON	ON
B000:0000 (704K)	OFF	ON	OFF	OFF	ON	ON
C000:0000 (768K)	OFF	OFF	ON	ON	ON	ON
D000:0000 (832K)	OFF	OFF	ON	OFF	ON	ON
E000:0000 (896K)	OFF	OFF	OFF	ON	ON	ON
F000:0000 (960K)	OFF	OFF	OFF	OFF	ON	ON
A000:0000 (640K)	OFF	ON	OFF	ON	ON	ON
A400:0000 (656K)	OFF	ON	OFF	ON	ON	OFF
A800:0000 (672K)	OFF	ON	OFF	ON	OFF	ON
AC00:0000 (688K)	OFF	ON	OFF	ON	OFF	OFF

Board Installation

Before installing the board, be sure that the power is off and that the power cord is removed from the PC!

Step 1 — Open the PC by removing the cover mounting screws on the rear of the system unit. Slide the cover of the system unit forward as far as possible without removing it from the system unit.

Step 2 — The board can be installed in any one of the available expansion slots on the system board. If you do not plan to use the remote switch, the left-most expansion slot will be the most convenient for reaching the switch located on the board's mounting bracket. Select an available expansion slot and remove the metal bracket from the back panel for that slot, using a small screwdriver. The metal bracket may be discarded, but be sure to save the retaining screw.

Step 3 — Align the board with the expansion slot and lower it until the edge connector is resting on the expansion slot receptacle on the system board. Press the board straight down until it seats in the expansion slot. Install the retaining screw through the board's bracket into the PC's back panel and tighten it.

Step 4 — Slide the cover of the system unit back over the machine and install the cover mounting screws.

Step 5 — If you plan to use the remote switch, install it now, while the power is still off. Remove the dummy plug from the phono jack mounted on the bracket and insert the phono plug that is connected to the remote switch.

If you do not plan to use the remote switch, leave the dummy plug in place. This will help prevent the accidental use of this jack for some other potentially dangerous use — such as the connection of a composite monitor.

Step 6 — Re-connect all peripherals and replace the power cord.

Step 7 — Boot the system. Install PERISCOPE by entering **PS** when the DOS prompt is displayed. If you changed the DIP switches, you'll need to specify the memory and/or port settings on the PERISCOPE command line. For example, if the memory was changed to C400:0000 and the port was changed to 304H, enter **PS /M:C400 /P:304**. See the next chapter for more information on the installation options. If an error occurs, see Appendix A for an explanation of the error.

Step 8 — After installing PERISCOPE, press the remote switch or the switch located on the board's bracket. The machine should display the message **We interrupt this program ...**, followed by PERISCOPE's command list. Enter **C** and press return to continue.

IV — Tutorial

This chapter takes you through a guided tour of PERISCOPE, using a simple assembly language program. This tutorial demonstrates most of PERISCOPE's debugging commands, but for more detailed information, you'll need to see Chapter V.

Before you can take the tutorial, you'll need to install the memory board (see Chapter III). To get started, place the PERISCOPE disk in drive A and make drive A the default disk drive. Then enter **PS /H** from the DOS prompt. This will load the resident portion of PERISCOPE into the protected memory and will also load the on-line help file. If you've changed the DIP switches on the board, be sure to enter the memory and/or port options on this line as well.

The program used in this tutorial is named SAMPLE.COM. This simple program displays the total and the available memory in the system. There are four files associated with this program. They are:

- SAMPLE.ASM — The source code for SAMPLE.COM
- SAMPLE.COM — The executable program
- SAMPLE.DEF — Contains record definitions for the PSP (Program Segment Prefix) and the FCB (File Control Block) and other records
- SAMPLE.MAP — The MAP file produced by the linker when the /M option is used

Before you start debugging SAMPLE.COM, use the DOS TYPE or PRINT command to print a listing of SAMPLE.ASM for reference. The program's mainline code starts at P010 and ends at P090. The mainline calls three procedures. The first procedure, P100, is called once to retrieve the total memory and the available memory. P200 is called twice to convert the memory size from hex to ASCII. Finally, P300 is called once to display the results.

Now that PERISCOPE is installed and you have a listing of SAMPLE.ASM for reference, start the program loader RUN by entering **RUN SAMPLE.COM** and pressing the return key.

RUN displays the address of the PSP and messages indicating that the address references and record definitions were loaded into the symbol and record tables, respectively. Then the display switches to the debugger screen and the first instruction of the program is displayed —

```
AX=0000  BX=0000  CX=008B  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=0C73  ES=0C73  SS=0C73  CS=0C73  IP=0100   NV UP EI PL ZR NA PE NC
                SAMPLE:
0C73:0100  EB35          JMP     P010
```

Registers BX and CX show the size of the program. Registers DS, ES, SS, and CS all show the PSP since this is a COM program. The actual number will vary, depending on the version of DOS and any memory-resident programs you're using.

Also, notice the symbols — the current instruction is labeled SAMPLE, since the name was defined as PUBLIC. The address of the jump is P010, not an offset. If you need to know the offset, you can find it in the hex display just after the address.

For help, enter **?** and press return. A command summary is displayed, from which you can see the possible commands. Now enter **? D** to get help on the display command.

To display the PSP, enter **D 0**. This shows the first 128 bytes of the Program Segment Prefix in byte format. For a more useful display, enter **DR 0 @PSP**. The record definition makes it much easier to see what's what in the PSP. Remember that you can add record definitions as you need them by editing the DEF file. To see the record definitions available, press F7 before entering anything after the debugger prompt.

To move to the next instruction, enter **T**. Now you're at the instruction labeled P010. Enter **U** to disassemble the next few lines —

```
                P010:
0C73:0137  E81700          CALL    P100
0C73:013A  A13301          MOV     AX,[TOTMEM]
0C73:013D  BF1001          MOV     DI,0110                    ; TMEMORY
0C73:0140  E82400          CALL    P200
0C73:0143  A13501          MOV     AX,[FREMEM]
0C73:0146  BF2C01          MOV     DI,012C                    ; AMEMORY
0C73:0149  E81B00          CALL    P200
0C73:014C  E83400          CALL    P300
                P090:
0C73:014F  CD20            INT     20
                P100:
0C73:0151  B106            MOV     CL,06
0C73:0153  BE0200          MOV     SI,0002
0C73:0156  8B04            MOV     AX,[SI]
```

Notice the two lines with the commented references to TMEMORY and AMEMORY. These instructions are ambiguous references to items in the symbol table. This situation occurs when a number is moved to a register. PERISCOPE cannot be sure that the references exist in the source program, so it displays the symbol as a comment. In this case, the instructions reference the symbols, but a situation where 100H is moved to a register would give a false reference to the program entry point, SAMPLE.

The search address command is used to search for address references. Enter **SA @P010 @P300 @P200** to search from P010 to P300 for references to the procedure P200. Also, try **SA @P010 @P300 @TOTMEM** to search the same range of memory for references to the data variable TOTMEM.

Before continuing, set a 'sticky' code breakpoint at the end of the program, P090. To do this enter **BC @P090** and press return.

To see the symbols available, press F8. Set a word breakpoint on the value of the variable TOTMEM changing from zero to any other value. To do this enter **BW @TOTMEM NE 0** and press return. To see the current breakpoints, enter **BA ?**. The result is —

```
BC  P090
BW  TOTMEM NE 0000
```

Now enter **GB** to execute the program with both of the breakpoints activated. Execution will stop at the instruction after the one that moves register AX into TOTMEM —

```
AX=0100  BX=0000  CX=0006  DX=0000  SP=FFFC  BP=0000  SI=0002  DI=0000
DS=0C73  ES=0C73  SS=0C73  CS=0C73  IP=015D   NV UP EI PL NZ NA PE NC
0C73:015D  8CCB           MOV      BX,CS
```

Now, clear the word breakpoint by entering **BW ***. Check the breakpoints by entering **BA ?**. Only the one code breakpoint should be present.

To see the current value of TOTMEM, enter **DW @TOTMEM L 2**. This is the total memory in K. To check the value, use the translate command. Enter **X nnnn**, where nnnn is the value of TOTMEM. The decimal value is displayed as the second field. Since the value of TOTMEM is still in register AX, **X AX** gives the same result.

Use the Jump command to trace through the next few instructions. Enter **J** and press return. Then press F4 to repeat the previous command. When you get to the RETurn, use the **T** command to return to the mainline code, since Jump does not work when the current instruction is a RET, IRET, or any form of JMP.

Clear the screen by entering **K** and pressing return. Then disassemble the number conversion routine by entering **U @P200 @P300**. To get back to the current instruction, enter **R** and press return.

To check the number conversion routine, P200, use the Jump command three times to get to the instruction after first call to P200. Display the converted value by entering **D @TOTAL** or **D @TMEMORY**. This value should agree with the converted value of TOTMEM displayed previously.

Since the 'sticky' code breakpoint for P090 is still in effect, enter **G** to go to P090. Alternately, if the sticky breakpoint did not exist, you could enter **G @P090**.

The remaining commands are not germane to this sample program, but we'll explore some of them here, using memory outside the program.

To display the interrupt vectors at the beginning of memory in double word format, enter **DD 0:0**.

To change the value of register CX, enter **R CX** and press return. When the colon prompt is displayed, enter **10** and press return. Alternately, you can do the same thing in one line by using a semi-colon for a pseudo carriage return — try entering **R CX;10**.

Display memory at offset 200H by entering **D DS:200**. Now clear this unused memory by using the fill command. Enter **F DS:200 L 80 ' '** to store 80H bytes of spaces. Now, enter **D 200** again —

```
0C73:0200  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
           * 0006 LINES OF 20 SKIPPED *
0C73:0270  20 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
```

See how the six lines of spaces in the middle of the display were omitted?

Use the enter command to modify memory — **E DS:240 'test'** enters the string 'test' starting at DS:240.

The search command is used to find a byte/string pattern. Enter **S 100 400 'test'** to search from offset 100H to offset 400H for the string 'test'. It will be found at offset 240.

The compare command is used to compare two blocks of memory. Enter **C 200 L CX 240** to compare the 10H bytes (the value of register CX) starting at offset 200H to the 10H bytes starting at offset 240H.

```
0C73:0200  20  74  0C73:0240
0C73:0201  20  65  0C73:0241
0C73:0202  20  73  0C73:0242
0C73:0203  20  74  0C73:0243
```

The first four bytes are different, since the first block contains spaces and the second block contains the string 'test'. The remaining twelve bytes are the same, so nothing is displayed for them.

To copy one block of memory to another, the move command is used. Enter **M DS:240 24F DS:200** to copy the contents of 240H through 24FH to 200H through 20FH. Use **D 200** to verify the move.

To perform hex arithmetic, enter **H** followed by a number, an operator (+, -, *, or /) and a second number and press return. For example, to subtract 20H from 10H, enter **H 10-20**.

To quit the debugger, enter **Q**. The command list is then displayed. Enter **R** to return to DOS or **C** to continue and then press return. Either response ends the debugging session.

V — PERISCOPE — The Debugger

This chapter describes the installation and use of the resident debugger, PERISCOPE (PS.COM). The following topics are covered:

- Installation options and defaults
- Analyzing record definition table size using RS.COM
- Analyzing symbol table size using TS.COM
- The command list (Boot, Continue, Debug, Return to DOS, Short boot)
- Keyboard usage
- Debugger parameters
- The debug commands

Installation Options

To load PERISCOPE, enter **PS** from the DOS prompt. This copies the debugger to the protected memory and verifies that the copy was successful. PERISCOPE has the following default values:

- The write-protected memory begins at C000:0000 and ends at C000:3FFF.
- The write-protect ports are 300H and 301H.
- One monitor is assumed.
- The original screen size is presumed to be 4K, which is sufficient for text mode only.

- The BIOS interrupt vectors used by PERISCOPE (9H, 10H, 16H, 17H, and 1CH) are set to their power-on values when PERISCOPE is used, and restored to the original values when you exit PERISCOPE.

- The record table size is presumed to be 1K.

- The symbol table size is presumed to be 1K.

- On-line help is not available.

The command-line options described below can be used to change the above defaults. All commands contain a slash (/) and a single letter mnemonic. Some of the commands include a number. If a number is used, it is always preceded by a colon (:). Note that the number is always in hexadecimal.

The command-line options are:

/A — Use an alternate debug screen. This option indicates that you have both a monochrome and a color monitor attached to the system via separate display adapters. The debugger uses the monitor that is not currently active when it is invoked, i.e. when the button is pressed or when a program is loaded with RUN.COM. If this option is used, the message **Alternate monitor will be used for display** is displayed when PERISCOPE is started.

When possible, use the monochrome monitor for the debugger display, since the monochrome display is about four times faster than the color display.

/D — Restore the original INT 13H vector before a short boot. This option is used with certain RAM disk software to re-point the diskette interrupt vector to BIOS before using the short boot option. It is needed only if the RAM disk software you use modifies the original interrupt 13H to point to memory that is corrupted by a short boot. If this option is used, the message **INT 13H will be reset when short boot option is used** is displayed when PERISCOPE is started.

/H — Install the on-line help into normal RAM. If this option is specified, the file PSHELP.TXT is loaded into RAM and on-line help will be

available from the resident debugger. The file must be in the default directory. If this option is used, the message **HELP file loaded into memory** is displayed when PERISCOPE is started.

The amount of memory required is the same as the size of the file. The file PSHELP.TXT is a normal ASCII text file. It can be edited as needed to add or remove help information. Be sure to leave the back-slashes and commands on a separate line and to end the file with a single back-slash.

/M:nnnn — Set the protected memory segment to something other than C000H. The four-digit hexadecimal number nnnn represents the segment to be used. Be sure that the segment used does not conflict with other memory installed in the system and that the desired segment is indicated by the DIP switches on the board. If you must change the default setting, be sure to carefully follow the memory switch setting procedures in Chapter III. If this option is used, the message **Protected memory starts at segment nnnn** is displayed when PERISCOPE is started.

/P:nnnn — Set the protected memory ports to something other than 300H and 301H. The one to four-digit hexadecimal number nnnn represents the lower of the two ports to be used. Be sure that the ports used do not conflict with other ports installed in the system and that the desired port is indicated by the DIP switches on the board. If you must change the default setting, be sure to carefully follow the port switch setting procedures in Chapter III. If this option is used, the message **Memory protection port is nnnn** is displayed when PERISCOPE is started.

The architecture of the 8088 supports 64K I/O ports (0 through FFFF), but the IBM PC and XT support only the first 1024 (3FF) of these ports, many of which are reserved. The high 6 bits are ignored, with the result that port 1200H is really 200H, etc.

/R:nn — Set the size of the record definition table to something other than 1K. This option is used when debugging programs with large DEF files and large resultant record tables. The one or two-digit hexadecimal number nn is the number of K desired. The number may be from 1K (/R:1) to 32K (/R:20). Remember that the input is in hex! If this option is used, the message **Record table size is nnH KB** is displayed when PERISCOPE is started. See the next section for instructions on using the RECORDSIZE program (RS.COM) that determines the record table size required for a DEF file.

/S:nn — Set the size of the original or saved screen buffer to something other than 4K. This option is used when debugging programs that use the color-graphics adapter. The one or two-digit hexadecimal number nn is the number of K desired. If you're using the standard color-graphics adapter, and need 16K, enter **/S:10**. Remember that the input is in hex! The maximum size allowable is 32K, using **/S:20**. Keep this number as small as possible, since each Trace or Go command has to copy this buffer twice. If this option is used, the message **Screen buffer size is nnH KB** is displayed when PERISCOPE is started.

/T:nn — Set the size of the symbol table to something other than 1K. This option is used when debugging programs with large MAP files and a large resultant symbol table. The one or two-digit hexadecimal number nn is the number of K desired. The number may be from 1K (/T:1) to 63K (/T:3F). Remember that the input is in hex! If this option is used, the message **Symbol table size is nnH KB** is displayed when PERISCOPE is started. See the next section for instructions on using the TABLESIZE program (TS.COM) that determines the symbol table size required for a MAP file.

/V:nn — Indicate a BIOS interrupt vector that is to be left alone. Normally, when PERISCOPE is activated, it temporarily resets the interrupt vectors it uses to the power-on default values. This is done to make PERISCOPE as dependable as possible. When Trace or Go is used, or when PERISCOPE is terminated, the interrupt vectors are restored to their previous values. If you have a situation where you want PERISCOPE to use your modified interrupt vectors while it is running, use the /V:nn option, where nn is the one or two digit hexadecimal interrupt number. The possible numbers are 9 (keyboard), 10 (video), 16 (keyboard I/O), 17 (printer), and 1C (timer control). Note that each vector must be entered separately. For example, to leave vectors 10 and 17 alone, use **/V:10 /V:17**. If this option is used, the message **Interrupt nnH will be preserved when PERISCOPE is active** is displayed when PERISCOPE is started. PERISCOPE changes the Ctrl-Break vector (1BH) and the DOS Fatal error vector (24H), but these changes cannot be overridden.

The command line can be entered in any combination of upper and lower case. No spaces are required between entries. Some examples follow:

PS /A /S:10 — Use two monitors, and reserve 16K to save the original (graphics) screen.

PS /M:A000 /P:31C — Use memory in the screen buffer area (A000:0000 to A000:3FFF) for the protected memory and use ports 31C and 31D for the write-protect ports. Note: Ports 31C and 31D are reserved for the IBM prototype card. Using ports that are reserved by IBM is usually a safe bet if you're not using the corresponding board. See the IBM Technical Reference Manual for a list of reserved ports. Also see the documentation for any non-IBM boards in your system.

PS /T:20 /V:10 /H — Reserve 32K for the symbol table, preserve the current INT 10H vector when PERISCOPE is active, and load the on-line help file.

The cumulative size of the original screen buffer (4 to 32K), the debug screen buffer (always 4K), the record definition table (1 to 32K), the symbol table buffer (1 to 63K), and the on-line help (same as the size of PSHELP.TXT) can be very large. If you're using a version of DOS prior to 2.00, the total size of these buffers must be 63K or less. For DOS 2.00 and later, the total size can be from 10K to 128K or more. These buffers are located in normal RAM using the terminate and stay resident function of DOS. The protected memory is used for the code and critical data areas of the resident debugger.

If any errors are encountered during the initialization process, PERISCOPE displays an error message and terminates. See Appendix A for a list of the error messages. It is possible to hang the system by specifying an invalid port or memory address or by setting the DIP switches incorrectly.

PERISCOPE should be installed via an AUTOEXEC.BAT file to insure its presence each time the system is booted. If you are not sure if PERISCOPE is installed, do not press the button — try running RUN .COM instead. If PERISCOPE is not installed, RUN will display an error message.

Record Table Size

The program RS.COM reads a DEF file containing record definitions and displays the number of record definitions found and the total record table size required for a given DEF file. To run this program, enter **RS progname** from the DOS prompt. The file extension is presumed to be **DEF** and the file is presumed to be in the current directory. The DEF file is presumed to be in the same format as the file SAMPLE.DEF.

A section of the SAMPLE.DEF file (excluding comments) is shown below.

```
\FCB
Drive,b,1
File,b,8
Extension,b,3
Block No,w,2
Rec Size,w,2
File Size,d,4
Date,w,2
Reserved,b,10
Rec No,b,1
Rel Rec No,d,4
```

The first line of the file defines a record name. The record name is limited to 16 characters and must be preceded by a back-slash. No embedded spaces are allowed in the record name.

Each of the following lines defines a field within the record. Each of the lines contains a field name, a field type, and a field length, separated by commas. The field name may be up to 10 characters long and may have embedded spaces. The field type must be B, D, or W, indicating byte, double word, or word formatting respectively. The field length is the total number of bytes required by the field. This number is in hexadecimal notation. If double word formatting is used, the length must be a multiple of four. If word formatting is used, the length must be a multiple of two. The length of any one field and the total length of the record may be from one to FFFFH. Each line may be commented using a semi-colon preceding the comments.

Symbol Table Size

The program TS.COM reads a MAP file as produced by the linker and displays the number of address references, number of line references, and the total symbol table size required for a given MAP file. To run this program, enter **TS progname** from the DOS prompt. The file extension is presumed to be **MAP** and the file is presumed to be in the current directory. The MAP file is presumed to be in the format as output by the Microsoft and IBM Link programs.

Microsoft and IBM have made some subtle changes in the format of the MAP file from time to time. If you encounter problems reading a MAP file, please contact us as soon as possible.

To generate address references in the MAP file, you'll need to specify both a MAP file and the **/M** option at link time. Entries are generated in the MAP file for names defined as PUBLIC by your compiler or assembler. For ASM programs, a PUBLIC statement is used to generate an address reference in the MAP file. For Lattice C programs, variables defined as STATIC and external references to other modules will generate address references in the MAP file. For Pascal programs, variables defined as PUBLIC and external references to other modules will generate address references in the MAP file.

Only the first 16 characters of a public name are used by PERISCOPE. Any characters beyond the 16-character limit are discarded. The programs TS.COM and RUN.COM read the first group of address entries in the MAP file — the one sorted by name. Any absolute references found in the MAP file are included, but are not relocated to the program's location.

To generate line references in the MAP file, you'll need to specify a MAP file and the **/L** option at link time. Not all compilers support the line number option. The ones that are known to support this option are: Lattice/Microsoft C, Microsoft Pascal, and Microsoft FORTRAN.

The line references generated by TS.COM and RUN.COM have a single-character alphabetic prefix, followed by the actual line number. The alphabetic prefix starts at **A** and is incremented for each module found in the MAP file. For example, line 10 of the first module is referenced as **A10**, and line 20 in the second module is referenced as **B20**.

The Command List

When you press the button to activate the resident portion of PERISCOPE or when you use **Q** to quit the debugger, PERISCOPE's command list is displayed. It prompts you as follows:

We interrupt this program ... Normal boot (B), Continue (C), Debug (D), Return to DOS (R), or Short boot (S)? _

The **We interrupt this program** message is replaced with **Parity error 1** or **Parity error 2** if a memory parity error occurred on the motherboard or in add-on RAM respectively. To choose one of the options, enter the appropriate letter and press return. Any invalid keystrokes cause the system to beep and the response area to be cleared.

The normal boot (B) option performs the same function as Alt-Ctrl-Del, clearing all of user memory and resetting the standard interrupt vectors. This option can be used when the system is hopelessly confused or when you suspect that a runaway program may have incorrectly modified some part of memory.

The continue (C) option returns control to the executing program after restoring the program's screen. If nothing is executing, i.e. the DOS prompt is displayed, control is returned to DOS.

The debug (D) option is used to enter the debugger. Once the debugger is active, a Quit command is used to return to the command list. When the debugger is active, the > character is used for a prompt. To get help, enter **?** followed by the command. If the help file was not loaded, use **?** with no arguments for a command summary.

The return to DOS (R) option is used to abandon execution of the current program and return to DOS. Note that any open files will not be closed — this can cause problems if the files have been updated. Also, any changes the program has made to interrupt vectors will not be backed out.

The program RUN.COM stores the current Program Segment Prefix (PSP) in PERISCOPE's data area so PERISCOPE knows where to terminate. If RUN.COM has not been run, an attempt to return to DOS returns an error, since PERISCOPE doesn't know where to find the PSP. When a batch file is being run, DOS moves the PSP up in memory a few paragraphs. Also, the DOS TYPE command copies the file to be typed

into the area normally used by the PSP. Other situations may alter the PSP, preventing use of the Return to DOS option.

The short boot (S) option is used to re-boot the system via interrupt 19H. This method preserves most of RAM, including the interrupt vectors. Some sections of memory in the first 48K are overwritten by the boot record and DOS. This can cause problems for some memory-resident programs, such as low-memory RAM disks, timer & keyboard handlers, etc.

If a program has changed interrupt vectors, problems may occur as well. For example, the short boot option should not be used with BASIC 2.00 or later since BASIC resets the timer vector to point to a location within itself. After a short boot, the code may still be there, but if you load a program, the code will be overlaid and the system will lock up.

Keyboard Usage

Since PERISCOPE uses DOS functions for the Load, Name, and Write commands only, not all of the DOS editing capabilities are available when you're debugging a program. This section describes the editing capabilities that are implemented in the debugger.

You may use the following keys to edit the previously entered command line:

F1 — copy one character from the previous command line into the current command line (same as DOS).

F3 — copy the remainder of the previous command line into the current command line. This key copies up to, but not including the carriage return (same as DOS).

F4 — same as F3, except that a carriage return is added at the end of the command line. For repetitive commands, you can use just one keystroke — F4.

Ins — places the current command line into insert mode (same as DOS).

Del — deletes a character from the previous command line (same as DOS).

Esc — cancel the current command line (same as DOS).

Other keys are defined as follows:

Ctrl-Break — cancels the current command and returns to the debug prompt.

Ctrl-PrtSc — toggles printer echo of screen output on and off (same as DOS).

Ctrl-S — suspend output until another key is pressed. This key combination is used to keep information from scrolling off the screen too quickly.

F7 — display the current record definitions as read from a DEF file when RUN is started. This key is ignored unless the cursor is at the beginning of a command line.

F8 — display the address and name of the symbol table entries as read from a MAP file when RUN is started. This key is ignored unless the cursor is at the beginning of a command line.

F10 — switches from the debug screen to the original screen if only one monitor is being used. If two monitors are used and the /A option was used to start PERISCOPE, this key has no effect. To return to the debugger screen from the original screen, press any key.

Semi-colon — This character is used as a pseudo carriage-return. Using it, you can enter multiple commands on one line. For example, if you're tracing through a program that requires repetitive Go commands, you could enter **G @PRINT_LINE;D @BUFFER;T** to go to the line labeled PRINT_LINE, display memory starting at BUFFER, and then trace one more line to be ready for the next Go command. After the line has been entered once, you can use F4 to repeat it.

Shift-PrtSc — prints the entire screen to the parallel printer (same as DOS).

Debugger Parameters

The debugger is command driven. Input may be entered in either upper or lower case. A space or comma can be used to delimit parameters within a command. A delimiter is required when a sub-function is omitted, after a symbol but before another entry, and between two numbers.

Each command requires at least a single-character mnemonic. All but a few commands require additional input. Brackets ([ ]) are used to indicate an optional entry. Ellipsis (...) are used to indicate a repetitive entry.

The various parameters used by the debugger are defined below, in alphabetical order.

<address> — The address of a memory location. The address is composed of a segment and an offset, separated by a colon. Alternately, registers can be used for either or both numbers, or a valid symbol can be used for both the segment and offset. For some commands, the segment may be omitted. Possible addresses include **1000:1234**, **DS:SI**, and **@PRINT_LINE**.

<arithmetic operator> — The arithmetic symbols +, -, *, and /, used for addition, subtraction, multiplication, and division, respectively.

<byte> — A one or two digit hexadecimal number from 0 to FF.

<decimal number> — A decimal number from 0 to 65,535. No punctuation is allowed.

<drive> — A single-digit number corresponding to a disk drive, where 0 equals drive A, 1 equals drive B, etc.

<flag> — A flag register. The possible values and two-character mnemonics are:

FLAG	SET (=1)	CLEAR (=0)
Overflow	OV	NV
Direction	DN (STD)	UP (CLD)
Interrupt	EI (STI)	DI (CLI)
Sign	NG	PL
Zero	ZR	NZ
Auxiliary carry	AC	NA
Parity	PE	PO
Carry	CY (STC)	NC (CLC)

<function> — The debugger command, such as D (display memory), or U (un-assemble).

<length> — The number of bytes affected by a command. This may be represented by **L nnnn** where nnnn is a hexadecimal number from 1 to FFFF. It may also be represented by a number following an address. In this case the length is calculated as the number plus one minus the offset. For example, **D CS:100 L 100** and **D CS:100 1FF** (1FF plus 1 minus 100) both have a length of 100H.

A register name may be substituted for the number in either format. The current value of the register is used for the number.

A symbol may also be used for the length argument. The segment associated with the symbol must be the same as the segment referenced in the preceding address and the offset must not be less than the offset referenced in the address.

<list> — A list of byte(s) and/or string(s) used with the enter, fill, and search commands. For example **03 'COMMAND COM' 12 34** is a list composed of a byte, a string, and two trailing bytes.

<name> — A string of characters that are copied as entered to the unformatted parameter area in the Program Segment Prefix.

<number> — A one to four digit hexadecimal number from 0 to FFFF. If a register name is used, its current value is substituted for the number.

<offset> — The one to four digit hexadecimal number or register representing the offset into the specified segment.

<port> — The one to four digit hexadecimal number associated with an I/O port.

<range> — An address and a length. For example **CS:100 L 100** and **0:0 FF** are ranges. Two symbols may be used, providing that they both reference the same segment and that the offset of the second symbol is greater than or equal to the offset of the first symbol.

<register> — A machine register. The possible registers are AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, and IP.

<sectors> — Two hexadecimal numbers representing the starting relative sector number and the total number of sectors (max 80H). The sector numbering scheme is the one used by DOS interrupts 25H and 26H.

 — A one to four digit hexadecimal number or register representing one of the four current 64K segment registers (Code, Data, Extra, or Stack).

<string> — A quoted list of ASCII characters. Either single or double quotes may be used to delimit the string. To enter a string containing an embedded quote, use the other form of a quote to delimit the string, or enter two quotes where the single embedded quote is desired.

<sub-function> — The mnemonic used with some commands. For example, to display memory in word format, enter **DW**, where **W** is the sub-function. The sub-function must follow the function immediately — no intervening spaces are allowed. This is necessary to differentiate between a sub-function and an address. For example, consider **DD** and **D D**. The first command displays data in double word format starting at the current segment and offset. The second command displays data in byte format starting at offset D in the current segment.

<symbol> — A name corresponding to an address or a record definition. Symbols are loaded from a MAP file when the corresponding program is loaded by RUN. On entry, a symbol is always preceded by @. For example, to disassemble memory starting at the symbol **PRINT_LINE**, enter **U @PRINT_LINE**.

Symbols are also used to reference record definitions read from a DEF file. For example, to display the FCB, enter **DR CS:5C @FCB**.

<test> — Used to compare two values. The possible tests are LT (less than), LE (less than or equal), EQ (equal), NE (not equal), GE (greater than or equal), and GT (greater than).

Command: Help

Syntax: ? [<function><sub-function>]

Description: This command displays the debugger commands by function and sub-function if the on-line help file has been loaded. If the help file is not available, a command summary is displayed.

Examples:

? displays a command summary.

? DD displays help for the display double word command if the on-line help file has been loaded.

Command: Display Breakpoints

Syntax: BA ?, BB ?, BC ?, BM ?, BR ?, or BW ?

Description: These commands display the current breakpoints. The question mark is required only when multiple breakpoint functions are being performed with one command.

The sub-function indicates the breakpoint group to be displayed as follows:

A — display all breakpoints, including Byte, Code, Memory, Register, and Word

B — display Byte breakpoints

C — display Code breakpoints

M — display Memory breakpoints

R — display Register breakpoints

W — display Word breakpoints

Examples:

Assume that a byte breakpoint had been set for the symbol LINE_COUNT equal to 38H and that a register breakpoint had been set for CX less than 5.

BB displays **BB LINE_COUNT EQ 38**.

BR ? displays **BR CX LT 0005**.

BA ? displays both of the above breakpoints.

Command: Clear Breakpoints

Syntax: BA *, BB *, BC *, BM *, BR *, or BW *

Description: These commands clear the current breakpoints.

The sub-function indicates the breakpoint group to be cleared as follows:

A — clear all breakpoints, including Byte, Code, Memory, Register, and Word

B — clear Byte breakpoints

C — clear Code breakpoints

M — clear Memory breakpoints

R — clear Register breakpoints

W — clear Word breakpoints

Examples:

Assume that a byte breakpoint had been set for the symbol LINE_COUNT equal to 38H and that a register breakpoint had been set for CX less than 5.

BB * clears the byte breakpoint.

BR * clears the register breakpoint.

BA * clears all breakpoints.

Command: Breakpoint on Byte

Syntax: BB <address> <test> <byte> [...]

Description: This command is used to set a breakpoint when a byte of memory meets a test.

Up to eight breakpoints may be set at one time. If a segment is not specified in the address, the current data segment is used. If any of the tests pass, the breakpoint is taken. To trace execution with this breakpoint enabled, the **GB** or **GT** command must be used. This breakpoint stops execution of a program on the instruction following the instruction that changed the specified byte of memory. Multiple breakpoints may be set on a single input line. The breakpoint clear and display functions may also be present on the line.

Examples:

BB @LINE_COUNT EQ 38 sets a byte breakpoint for the memory location corresponding to LINE_COUNT.

BB * DS:123 GT F0 ? clears all byte breakpoints, sets one, and then displays the byte breakpoint.

Command: Breakpoint on Code

Syntax: BC <address> [...]

Description: This command is used to set a breakpoint when an instruction is executed.

It performs the same function as addresses entered after the Go command, except that these breakpoints are remembered or 'sticky'. If a segment is not specified in the address, CS is presumed. Either the **G**, **GB**, or **GT** command may be used to enable code breakpoints. This breakpoint stops execution of a program before the instruction at the specified address is executed. Multiple breakpoints may be set on a single input line. The breakpoint clear and display functions may also be present on the line. See the Go command for more information.

Examples:

BC @PRINT_LINE sets a code breakpoint for the memory location corresponding to PRINT_LINE.

BC * CS:123 ? clears all code breakpoints, sets one, and then displays the code breakpoint.

Command: Breakpoint on Memory

Syntax: BM <address> <address> R and/or W [...]

Description: This command is used to set a breakpoint when a range of memory will be read and/or written as the result of an instruction.

The two addresses may have different segments, but the second address must be higher in memory than the first address. If a segment is not specified in the address, the current data segment is used. Up to eight breakpoints may be set at one time. A breakpoint will occur only if a read or write starts in the specified range. If any of the tests pass, the breakpoint is taken. To trace execution with this breakpoint enabled, the **GB** or **GT** command must be used. This breakpoint stops execution of a program on the instruction that will read or write the specified range of memory. Multiple breakpoints may be set on a single input line. The breakpoint clear and display functions may also be present on the line.

Examples:

BM @DATASTART @DATAEND W sets a memory breakpoint for memory from DATASTART thru DATAEND. Any instructions that will write to this range of memory cause a breakpoint to be taken, before the instruction is executed.

BM * SS:SP SS:FFFF RW ? clears all memory breakpoints, sets a breakpoint to trap any reads or writes to the memory from SS:SP (the current stack position) to SS:FFFF (the top of the stack segment), and displays the memory breakpoint.

Command: Breakpoint on Register

Syntax: BR <register> <test> <number> [...]

Description: This command is used to set a breakpoint when a register meets a test.

Up to one test per register may be set at one time. If any of the tests pass, the breakpoint is taken. To trace execution with this breakpoint enabled, the **GB** or **GT** command must be used. This breakpoint stops execution of a program on the instruction following the instruction that changed the specified register. Multiple breakpoints may be set on a single input line. The breakpoint clear and display functions may also be present on the line.

Examples:

BR CX EQ 0123 sets a breakpoint when register CX is equal to 123H.

BR * ES NE DS ? clears all register breakpoints, sets one, and then displays it. Note that DS is used for its current value only.

Command: Breakpoint on Word

Syntax: BW <address> <test> <number> [...]

Description: This command is used to set a breakpoint when a word of memory meets a test.

Up to eight breakpoints may be set at one time. If a segment is not specified in the address, the current data segment is used. If any of the tests pass, the breakpoint is taken. To trace execution with this breakpoint enabled, the **GB** or **GT** command must be used. This breakpoint stops execution of a program on the instruction following the instruction that changed the specified word of memory. Multiple breakpoints may be set on a single input line. The breakpoint clear and display functions may also be present on the line.

Examples:

BW @CHAR_COUNT EQ 1234 sets a word breakpoint for the memory location corresponding to CHAR_COUNT.

BW * DS:123 GT SI ? clears all word breakpoints, sets one, and then displays it. Note that SI is used for its current value only.

Command: Compare

Syntax: C <range> <address>

Description: This command is used to compare two blocks of memory a byte at a time.

If any differences are found, the address and value of the first byte and the value and address of the second byte is displayed. Nothing is displayed for bytes that match. Since this command accepts two addresses as input, the two blocks of memory may be in different segments. If no segment is input, the current data segment is used. The length parameter indicates how much memory is to be compared.

Assume that you want to compare memory location 3000:0000 with 3000:0010 for 8 bytes. Enter **C 3000:0 L 8 3000:10**. The result might be:

```
3000:0000  88  00  3000:0010
3000:0001  02  66  3000:0011
3000:0003  04  27  3000:0013
```

The above display shows three bytes that were different. Each line shows the first address, the value of the first address, the value of the second address, and the second address. Since the other five lines were not displayed, the values of these bytes were the same.

Examples:

C DS:SI L 100 ES:DI compares 100H bytes starting at DS:SI with 100H bytes starting at ES:DI.

C 123 L CX 456 compares memory starting at DS:123 with memory starting at DS:456. The number of bytes compared is the current value of register CX.

C @FCB1 L 25 @FCB2 compares memory starting at the symbol FCB1 with memory starting at the symbol FCB2 for 25H bytes.

Command: Display using Byte format

Syntax: D [<range>] or DB [<range>]

Description: This command is used to display a block of memory in hex and ASCII.

Each line of the display shows the starting segment and offset, up to 16 bytes, and their ASCII representation.

A dash is displayed between the eighth and ninth bytes for readability. If a display is not started on a paragraph boundary (i.e. the memory address is not evenly divisible by 16), a short line is displayed for the first line. Similarly, if the display does not end on a paragraph boundary, the last line will be a short line.

For the ASCII display, the high-order bit is ignored, i.e. a byte whose value is greater than 80H has 80H (128) subtracted from it before being displayed. Also, any bytes from zero to 1FH are displayed as a period.

For example, if you enter **D 0:0 L 20** or **DB0:0 1F** the display might look like this:

```
0000:0000  01 15 BF 00 5C 09 F0 BF-5C 09 F0 BF 55 09 F0 BF   ..?.\.p?\.p?U.p?
0000:0010  86 00 60 00 54 FF 00 F0-00 00 00 00 00 00 00 00   ..`.T..p........
```

If one or more lines in the middle of the display are found to be 16 occurrences of the same number, the line(s) are suppressed and a message of the form *** NNNN LINES OF XX SKIPPED *** is displayed in place of the line(s). NNNN is the number (in hex) of lines skipped and XX is the byte value found in all bytes in all of the skipped lines.

The syntax for this command is very flexible. If you enter **D**, it displays memory starting where the last **D** command left off. Note that the commands G, J, R, and T reset the starting point to DS:100. If you enter **D <number>** the number is presumed to be an offset, the segment is presumed to be DS, and the length is presumed to be 80H. If you enter **D <number> <length>** the number is presumed to be an offset, and the segment is presumed to be DS.

Examples:

D displays memory starting where the last **D** command left off, except if a G, J, R, or T command has been used.

D @LINE_COUNT L 1 displays the byte at the symbol LINE_COUNT.

DB ES:DI displays memory starting at ES:DI for a length of 80H.

Command: Display using Double word format

Syntax: DD [<range>]

Description: This command is used to display a block of memory in double word format.

This format is useful for examining data that is stored as a word offset followed by a word segment. Each line of the display shows the starting segment and offset and up to 4 pairs of segments and offsets. If the number of bytes displayed is not evenly divisible by 16, the last line will be a short line.

For example, if you enter **DD 0:0 L 20** or **DD0:0 1F** the display might look like this:

```
0000:0000  00BF:1501  BFF0:095C  BFF0:095C  BFF0:0955
0000:0010  0060:0086  F000:FF54  0000:0000  0000:0000
```

If one or more lines in the middle of the display are found to be 16 occurrences of the same number, the line(s) are suppressed and a message of the form *** NNNN LINES OF XX SKIPPED *** is displayed in place of the line(s). NNNN is the number (in hex) of lines skipped and XX is the byte value found in all bytes in all of the skipped lines.

The syntax for this command is very flexible. If you enter **DD**, it displays memory starting where the last display command left off. Note that the commands G, J, R, and T reset the starting point to DS:100. If you enter **DD <number>** the number is presumed to be an offset, the segment is presumed to be DS, and the length is presumed to be 80H. If you enter **DD <number> <length>** the number is presumed to be an offset, and the segment is presumed to be DS.

Examples:

DD displays memory starting where the last display command left off, except if a G, J, R, or T command has been used.

DD 0:0 L 20 displays the interrupt vectors 0 through 7.

DD @VECTORLIST displays memory starting at the symbol VECTORLIST.

Command: Display using Record format

Syntax: DR <address> <symbol>

Description: This command is used to display a block of memory by fields as defined by a record definition.

This format is useful for examining data that is part of a record, such as the PSP (Program Segment Prefix) or a FCB (File Control Block). Each line of the display shows a field name and the data for the field in byte, word, or double word format. Any area of memory can be displayed using any record definition.

To define a record format, a file of the name xxxx.DEF, where xxxx is the name of the program being debugged, must be present on the same drive as the program. The program loader, RUN.COM, reads this DEF file when loading the program and loads the record definitions into the record table. You can add record definitions to the file as needed by a specific program. See the sample file SAMPLE.DEF and the formatting discussion in the section on the Record size program (RS.COM).

The following definition of the PSP record is from the file SAMPLE.DEF.

```
\PSP
Int 20,b,2
Top Mem,w,2
Reserved,b,1
Long Call,b,1
DOS Func,d,4
Terminate,d,4
Ctrl-Break,d,4
Error,d,4
DOS Use,b,16
Environ,w,2
DOS Use,b,2e
PSP1,b,10
PSP2,b,14
```

Assuming that the definition for the PSP record has been loaded, enter **DR CS:0 @PSP** to get a display similar to the following:

```
Int 20      CD 20                                                   M
Top Mem     27C0
Reserved    00                                                      .
Long Call   9A                                                      .
DOS Func    F01D:FEF0
Terminate   02E3:013A
Ctrl-Break  02E3:0107
Error       02E3:0196
DOS Use     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00      ................
            00 00 00 00 00 00                                       ......
Environ     0000
DOS Use     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00      ................
            * 0001 LINES OF 00 SKIPPED *
            00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00         ................
PSP1        00 20 20 20 20 20 20 20 20 20 20 20 00 00 00 00 00      .           ....
PSP2        00 20 20 20 20 20 20 20 20 20 20 20 00 00 00 00 00      .           ....
            00 00 00 00                                             ....
```

If one or more lines in the middle of the display are found to be 16 occurrences of the same number, the line(s) are suppressed and a message of the form *** NNNN LINES OF XX SKIPPED *** is displayed in place of the line(s). NNNN is the number (in hex) of lines skipped and XX is the byte value found in all bytes in all of the skipped lines.

The syntax for this command is less flexible than the other display commands. You must enter an address and a record name. The address must have an offset and must not include a length.

Examples:

Assuming that the records PSP and FCB are defined (as in the file SAMPLE.DEF).

DR CS:0 @PSP displays the PSP, using memory starting at CS:0.

DR CS:5C @FCB displays the first FCB in the PSP, which starts at CS:5C.

DR @FCB1 @FCB displays the FCB starting at the address referenced by the symbol FCB1. Note that the symbol table is used for the first symbol and the record definition table is used for the second symbol.

Command: Display using Word format

Syntax: DW [<range>]

Description: This command is used to display a block of memory in word format.

This format is useful for examining data that is stored as words as opposed to bytes. It reverses out the 'back words' style of storage used on the 8088. Each line of the display shows the starting segment and offset and up to 8 words. If the number of bytes displayed is not evenly divisible by 16, the last line will be a short line.

For example, if you enter **DW 0:0 L 20** or **DW0:0 1F** the display might look like this:

```
0000:0000  1501  00BF  095C  BFF0  095C  BFF0  0955  BFF0
0000:0010  0086  0060  FF54  F000  0000  0000  0000  0000
```

If one or more lines in the middle of the display are found to be 16 occurrences of the same number, the line(s) are suppressed and a message of the form *** NNNN LINES OF XX SKIPPED *** is displayed in place of the line(s). NNNN is the number (in hex) of lines skipped and XX is the byte value found in all bytes in all of the skipped lines.

The syntax for this command is very flexible. If you enter **DW**, it displays memory starting where the last display command left off. Note that the commands G, J, R, and T reset the starting point to DS:100. If you enter **DW <number>** the number is presumed to be an offset, the segment is presumed to be DS, and the length is presumed to be 80H. If you enter **DW <number> <length>** the number is presumed to be an offset, and the segment is presumed to be DS.

Examples:

DW displays memory starting where the last display command left off, except if a G, J, R, or T command has been used.

DW SS:SP FFFF displays the stack from SS:SP to the top of the stack segment.

DW @POINTER displays memory starting at the symbol POINTER.

Command: Enter

Syntax: E <address> [<list>]

Description: This command is used to modify memory.

The segment and offset must be specified for the address, since this command modifies memory.

If the optional list is present, the specified memory is modified and the command terminates. If the list is not present, an interactive mode is started. This mode allows you to examine and optionally modify individual bytes starting at the specified address.

For example, if you enter **E 2000:123** and press return, the interactive mode is started. The program displays the address and the current value of the byte as **2000:0123 xx.**, where xx is the current value. To modify this value, enter the hex number (0 through FF). Any invalid input, such as **G9** or too many digits is not echoed.

Press the space bar to skip to the next byte. Press the hyphen key to back up one byte. The backspace key is used to discard a single digit. Use the return key to terminate the interactive mode. Note that the interactive mode is not compatible with the multiple command capability of PERISCOPE — i.e. you cannot use semi-colons to 'stack' multiple commands on one line.

When moving forward with the space bar, a new line is started when the address is evenly divisible by eight. When moving backward with the hyphen key, each address is on a new line.

Examples:

E CS:5C 0 'FILENAMEEXT' modifies the value of CS:5C through CS:67 to contain a binary zero and the string 'FILENAMEEXT'.

E 404:100 starts the interactive mode and displays **0404:100 80.**. To change this value to 88H, type 88. To display the next byte, press the space bar. To change the byte at offset 104 to 0, enter 0 when the byte is displayed. To back up to offset 102, press the hyphen key as many times as needed to get back to it. When you've finished your changes, press the return key.

Command: Enter Symbol

Syntax: ES <address> <symbol>

Description: This command is used to define or redefine symbol table entries.

A segment and offset must be specified for the address. The symbol name must 16 characters or less and must be preceded by @. The symbol table is searched for a symbol of the same name. If an existing symbol is found, the segment and offset associated with it are updated. If no match is found, a new symbol is added at the end of the symbol table.

Examples:

ES CS:100 @START defines a symbol named START to have a segment equal to the current value of CS and an offset of 100H.

ES ES:DI @OUTDATA defines a symbol named OUTDATA to have a segment and offset equal to the current values of ES and DI, respectively.

Command: Fill

Syntax: F <range> <list>

Description: This command is used to fill a block of memory with a byte/string pattern.

A segment and offset must be specified for the address, since this command modifies memory. The length specifies the number of bytes to be affected. The list is the pattern that is copied into the specified range of memory. If the length of the list is less than the length of the range specified, the list is copied as many times as needed to fill the range. Conversely, if the length of the list is greater than the length of the range, the extra bytes are not copied.

Examples:

F ES:0 L 1000 0 writes binary zeroes to memory starting at ES:0 for a length of 1000H bytes.

F DS:SI L CX 'test' writes the string 'test' to memory starting at DS:SI. If CX is 3, only 'tes' is copied. If CX is 8, 'test' is copied exactly two times, etc.

F @ARRAY @ENDARRAY 0 zeroes memory from the symbol ARRAY up to and including the symbol ENDARRAY.

Command: Go

Syntax: G [<address>] [...]

Description: The Go command is used to set optional temporary code breakpoints and execute the program being debugged.

If any addresses are specified on the command line, the byte at each of the addresses is replaced with a CCH, the single-byte breakpoint. When control is returned to PERISCOPE via any method, the original byte is restored. The addresses entered on the command line are referred to as temporary code breakpoints. Up to four of these breakpoints may be used. If the address does not contain a segment, the current code segment is used.

To set 'sticky' code breakpoints, use the **BC** command described above. This method allows you to set up to 16 sticky code breakpoints.

If **G** with no addresses is entered, the 'sticky' breakpoints, if any, are used. If there are no 'sticky' breakpoints, program execution continues until the button is pressed.

You cannot set code breakpoints in ROM — code breakpoints require that PERISCOPE be able to exchange the original byte with CCH before starting the Go.

If a Go command is entered for the same address as the current instruction, the Go is performed, but nothing will have happened. The reason is that the instruction shown is the one about to be executed. If you go until the current instruction is reached, nothing will have been executed. If you need to trace through a loop using the same address each time, try combining the Go command with a Trace command. Assume that you want to monitor the instruction at address 123. You'd use **G 123;T** to go until CS:123 was reached, and then trace down to the next instruction. Then, by pressing F4, you can perform the Go/Trace combination as many times as needed.

Examples:

All of the examples below set the 'sticky' code breakpoints, if any.

G @PRINT_LINE sets a temporary code breakpoint at the address equal to the symbol PRINT_LINE and starts execution of the program.

G FF00:0000 returns an error since the address is in ROM.

G begins execution of the program with no temporary code breakpoints.

G 123 sets a temporary code breakpoint at CS:123.

Command: Go using Breakpoints

Syntax: GB [<address>] [...]

Description: The Go using Breakpoint command is the same as the normal Go command, except that it also sets the non-code or 'monitor' breakpoints.

These breakpoints are byte (BB), memory (BM), register (BR), and word (BW). Using these breakpoints puts the system into a mode where every instruction executed by your program is analyzed to see if a breakpoint has been reached. This analysis can slow down the execution by a factor of 100 or more, but in many cases is the only way to find an elusive bug.

This command differs from the **GT** command in that it does not trace the execution of software interrupts performed during the execution of the command.

For temporary and 'sticky' code breakpoints, this command performs in the same fashion as the Go command described above.

Examples:

All of the examples below set the 'monitor' breakpoints as well as the 'sticky' code breakpoints, if any.

GB @PRINT_LINE @NEW_PAGE sets temporary code breakpoints at the addresses equal to the symbols PRINT_LINE and NEW_PAGE.

GB begins execution of the program with no temporary code breakpoints — only 'sticky' and 'monitor' breakpoints, if any.

GB ES:456 sets a temporary code breakpoint at ES:456.

Command: Go using Trace

Syntax: GT [<address>] [...]

Description: The Go using Trace command is the same as the Go using Breakpoints command, except that it forces the tracing of software interrupts called from your program.

This command differs from the **GB** command in that it traces the execution of software interrupts performed during the execution of the command. Since the **GT** command monitors every instruction executed, it is slower than the **GB** command.

See the **GB** command for more information.

Command: Hex arithmetic

Syntax: H <number> <arithmetic operator> <number>

Description: This command is used to perform hexadecimal arithmetic.

Addition, subtraction, multiplication, and division are available. The standard symbols are used for each function. The numbers must be in hex and may be from one to four bytes. If a register name is entered in place of one of the numbers, its current value is used for the number.

Multiplication returns two words separated by spaces. The first word is the high-order part. Division returns two words separated by the letter 'R'. The first word is the quotient and the second is the remainder.

Examples:

H 1234 + 123 gives an answer of 1357

H 1234-123 gives an answer of 1111

H 1234/123 gives an answer of 0010 R 0004

H 1234*123 gives an answer of 0014 B11C

H DI-SI displays the result of subtracting the current value of SI from the current value of DI

Command: Input

Syntax: I <port>

Description: This command is used to read an I/O port.

The port number may be from zero to FFFF, although the IBM PC only supports ports from zero to 3FF — any larger number is effectively ANDed with 3FF. The byte value retrieved by reading the port is displayed on the line following the command.

Examples:

I 100 performs a read of port 100H and displays the byte input.

I DX performs a read of the port indicated by register DX and displays the byte input.

Command: Jump

Syntax: J

Description: This command is used as a shorthand form of Go — to jump to the next instruction.

It enables you to skip over the current instruction and go to the next instruction on the same level. It is used to skip over instructions that will return to the next instruction, such as CALL and INT. It is also useful for quickly moving through REP prefixes and LOOPs.

This command performs the same function as a temporary code breakpoint set on the next instruction — the difference is that you don't have to stop and compute the address and then enter a Go command — Jump does it for you.

There are two conditions under which this command does not work. The first situation is when you're tracing ROM — no code breakpoints can be used, since you can't write to ROM. The second condition is when the current instruction is any form of a RET, IRET, or JMP (including conditional jumps — whether or not they'll be taken). Since these instructions transfer control away from the next instruction, the Jump command cannot be used.

Generally speaking, it is safe to use this command in place of the Trace command. There are some cases that present a problem, however. One possibility is a LOOP instruction that passes control downwards rather than upwards. Another situation is a CALL or INT that does not return control to the next instruction.

Examples:

Assume that the current instruction is **INT 21**. Enter **J** to place a temporary code breakpoint at the instruction after the **INT 21** and automatically perform a Go command.

Assume that the current instruction is **RET**. Enter **J** and error 19 occurs — setting a code breakpoint at the instruction after a RET does not skip to the next instruction.

Command: Klear

Syntax: K

Description: This command clears the debugger screen. It has no arguments.

Example:

K clears the screen.

Command: Load Absolute disk sectors

Syntax: LA <address> <drive> <sectors>

Description: This command is used to load absolute disk sectors into memory.

The segment defaults to CS if no segment is specified in the address. The drive is a single digit number indicating the disk drive (0 = A, 1 = B, etc.). The sectors parameter is the starting sector number and the number of sectors to be read. The maximum number of sectors that can be read in one operation is 80H.

To use this command, DOS must not be active. See the description of the Name command for more information. This command uses DOS interrupt 25H. See the DOS manual for information on the numbering of the absolute disk sectors.

Examples:

LA DS:100 0 10 20 loads data into memory starting at DS:100 from drive A, starting at sector number 10H for 20H sectors.

LA 100 1 0 4 loads data into memory starting at CS:100 from drive B, starting at sector 0 for 4 sectors.

Command: Load File from disk

Syntax: LF [<address>]

Description: This command is used to load a file from disk into memory.

The optional address specifies where the file is to be loaded. If the address is not specified, CS:100 is used. To use this command, DOS must not be active. See the description of the Name command for more information. Before this command can be used the Name command must be used to specify a file name.

This command can be used to load any type of file into memory. After the file has been loaded, BX and CX indicate the size of the file in bytes. After the file is loaded into memory no other processing occurs — EXE files are not relocated or stripped of their headers. RUN.COM should generally be used to load and execute a program, since it loads the symbol table and performs relocation for EXE files. This command is useful for loading a file into memory for examination or modification.

Examples:

LF DS:1000 loads the file defined by a Name command into memory starting at DS:1000.

LF loads the file defined by a Name command into memory starting at CS:100.

Command: Move

Syntax: M <range> <address>

Description: This command is used to copy a block of memory to another location in memory.

The segment and offset must be specified for both addresses, since this command modifies memory. If the source block and target block overlap, the move into the target block is performed without loss of data. The source segment and target segment may be different.

Examples:

M 1000:0 L 100 1000:80 copies 100H bytes from the source block (1000:0 to 1000:FF) to the target block (1000:80 to 1000:17F). Since the source and target blocks overlap by 80H bytes, the move copies memory starting at 1000:FF and works down, so that the target block is an exact copy of the source block when the move command was started.

M 1000:80 L 100 1000:0 copies 100H bytes from the source block (1000:80 to 1000:17F) to the target block (1000:0 to 1000:FF). Since the source and target blocks overlap and the source is higher than the target, the move copies memory starting at 1000:80 and works up.

M DS:SI L CX ES:DI copies CX bytes from the source block (DS:SI) to the target block (ES:DI), where all values are the current contents of the respective registers.

Command: Name

Syntax: N <name>

Description: This command is used to enter data into the PSP for disk I/O.

The name specified after the leading N is copied to the unformatted parameter area in the Program Segment Prefix (PSP), starting at CS:80H. This command copies all data entered after the N until a carriage return is found — it ignores the use of a semi-colon for entering multiple commands on one line.

After the name is copied into CS:80H, the DOS parsing function is used to parse the first two file names in the command line into the File Control Blocks (FCBs) at CS:5CH and CS:6CH. If an invalid drive id is found on a file, a message is generated and register AL or AH is set to FF, indicating the first or second file, respectively.

Since the Name, Load, and Write commands use DOS calls, a check must be performed to be sure that DOS is not currently active. This is done by checking a flag that is set by RUN.COM and cleared when the button is pressed. For any of these three commands, this flag must be set, meaning that the button has not been pressed since RUN.COM was last run. Also, the current code segment must be the same as the PSP when RUN transferred control to PERISCOPE and the first two bytes of the PSP (CS:0 and CS:1) must be CDH and 20H. This is necessary to validate the memory size which is read from CS:2 and CS:3.

Examples:

N C:COMMAND.COM copies the command (except the N) to the unformatted parameter area at CS:80H and then parses the file name into the FCB at CS:5CH.

N FILE1,FILE2/N copies the command (except the leading N) to the unformatted parameter area at CS:80H and then parses the file names into the FCBs at CS:5CH and CS:6CH.

Command: Output

Syntax: O <port> <byte>

Description: This command is used to output a byte to an I/O port.

The port number may be from zero to FFFF, although the IBM PC only supports ports from zero to 3FF — any larger number is effectively ANDed with 3FF. The byte value output to the port may be from zero to FF.

Examples:

O 100 FF outputs FFH to port 100H.

O DX 12 outputs 12H to the port indicated by register DX.

O DX AX returns an error since register AX represents a word — only bytes can be output via PERISCOPE.

Command: Quit

Syntax: Q [<sub-function>]

Description: This command is used to exit the debugger and return to PERISCOPE's command list.

The optional sub-function is used to pre-answer the command list prompt. The possible combinations are **QB**, **QC**, **QD**, **QR**, and **QS** to quit and Boot, Continue, Debug, Return to DOS, or perform a Short boot, respectively.

Examples:

Q exits the debugger and displays the **We interrupt this program** prompt and the command list.

QC exits the debugger and continues execution of the interrupted program.

QS exits the debugger and performs a short boot.

Command: Register

Syntax: R [<register>] or [F]

Description: This command is used to display and/or modify the current values of the registers and flags.

If you enter **R** and press return, the current values of the registers and flags are displayed. If the current instruction performs a memory read and/or write, the effective address of the read/write is displayed, along with the current value of memory at the effective address(es). Finally, the current instruction is disassembled. The default segments and offsets for the **D** and **U** commands are reset to DS:100 and CS:IP respectively.

Some examples are shown below:

```
Example 1:

AX=007F  BX=0034  CX=0000  DX=0000  SP=1724  BP=00A0  SI=0F1E  DI=1560
DS=0040  ES=00BF  SS=00BF  CS=F000  IP=E850   NV UP DI PL ZR NA PE NC
F000:E850  74F3          JZ      E845                          ; jump


Example 2:

AX=0000  BX=0000  CX=0100  DX=0001  SP=FFFD  BP=0000  SI=0000  DI=0000
DS=063A  ES=063A  SS=063A  CS=063A  IP=010E   NV UP EI PL ZR NA PE NC
                                    WR DS:0131 = 0000
063A:010E  891E3101      MOV     [FILE_OFFSET],BX


Example 3:

AX=0000  BX=0000  CX=0100  DX=0001  SP=FFFB  BP=0000  SI=0000  DI=0000
DS=063A  ES=063A  SS=063A  CS=063A  IP=01AD   NV UP EI PL ZR NA PE NC
                P565:
063A:01AD  BF2F01        MOV     DI,012F                       ; FILE_SEGMENT
```

In all of the above examples, the first two lines display the current values of the thirteen registers and the eight flags. See the table below for an explanation of the flag mnemonics.

In Example 2, the third line shows that the current instruction performs a write to the word at DS:0131 and that the current value of the word is zero. If the instruction were to read memory, the left-hand side of line three would show that information. The evaluation of the effective address of memory reads and writes shows the effect of any and all memory access before the execution of the instruction.

In Example 3, the third line shows **P565**, the name of the current address from the symbol table. This line is present only when CS:IP exactly matches an entry in the symbol table.

The last line of each of the examples shows the disassembled instruction. The address of the instruction (CS:IP) is shown at the left, followed by the one to six bytes that make up the instruction, and the instruction itself.

If the current instruction is a conditional jump (see Example 1), the jump is evaluated based on the current flag settings as **jump** or **no jump**, meaning that the jump will or will not be taken, respectively.

If an address referenced by an instruction is found in the symbol table, the symbol name is substituted for the offset (see Example 2). If an ambiguous reference to an address is made, the symbol name is shown at the end of the disassembled instruction as a comment (see Example 3). This indicates that the symbol may or may not have been used in the original instruction. The most common ambiguous reference is generated by a move of an offset to a register, such as a **MOV DI,OFFSET FILE_SEGMENT**.

An address must match exactly for the symbol to be found. The current value of the segment used by the instruction (explicit or implicit) must match the segment in the symbol table. The offset used by the instruction must also match the offset in the symbol table.

To modify a register, enter **R <register>**. PERISCOPE displays the current value of the register, followed by a colon. If you enter a one to four digit hex number or another register name and press return, the register is changed. If you press return without entering a number, the register is not changed. The valid register names are AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, and IP.

To modify a flag, enter **R F**. PERISCOPE displays the current values of the flags (see the table below) followed by a hyphen. To change the flags, enter the desired mnemonics and press return. If you press return without entering any flag mnemonics, no flags are changed. The flags may be entered in any order, in upper or lower case, and with or without spaces between the entries.

FLAG	SET (=1)	CLEAR (=0)
Overflow	OV	NV
Direction	DN (STD)	UP (CLD)
Interrupt	EI (STI)	DI (CLI)
Sign	NG	PL
Zero	ZR	NZ
Auxiliary carry	AC	NA
Parity	PE	PO
Carry	CY (STC)	NC (CLC)

Examples:

R displays all registers and flags, the effective address for reads and/or writes, and disassembles the current instruction.

R AX displays the current value of register AX and prompts you for the new value. Press return to leave the register unchanged, or enter a one to four digit hex number and press return to change the register.

R AX;CX uses the multiple-command feature of PERISCOPE to change the value of register AX to the current value of register CX.

R F displays the current flags, followed by a hyphen. If you want to change the zero flag from NZ to ZR, enter **ZR** and press return.

Command: Search

Syntax: S <range> <list>

Description: This command is used to search memory for a byte/string pattern.

The block of memory specified by the range is searched for the pattern specified by the list. If a match is found, the starting address of the match is displayed and the search for matches continues at the next byte. If no matches are found, nothing is displayed. If no segment is specified in the address, the current data segment is used.

Examples:

S CS:IP L 200 CD 21 searches memory from the current instruction (CS:IP) for 200H bytes for the pattern CD 21. Any matches are displayed in segment:offset format.

S @PRINT_LINE L 50 C 'Page' searches 50H bytes starting at the address of the symbol PRINT_LINE for the byte 0CH followed by the string 'Page'.

Command: Search for Address reference

Syntax: SA <range> <address>

Description: This command is used to search memory for references to a specified address.

This command can be thought of as a disassembly that only shows instructions that reference an address of interest. To use it, specify an address range that is to be searched and the address reference that is to be searched for.

You can use this command to find JMPs and CALLs to a procedure or to find locations in your program where a data variable is accessed. Any instruction that references the specified address is displayed.

Examples:

SA CS:100 L 200 @P200 searches from CS:100 for 200H bytes for any references to the address represented by the symbol P200.

SA @PSTART @PEND DS:0 searches from the address represented by PSTART through the address represented by PEND for references to DS:0.

Command: Trace

Syntax: T [<number>]

Description: This command is used to execute the current program one instruction at a time.

If the optional number is not entered, one instruction is executed and control is returned to PERISCOPE. If the number is entered, that number of instructions are executed. For each trace the sequence of events is: the debugger screen is saved, the original program screen is restored, the instruction is executed, the program screen is saved, the debugger screen is restored, and the Register command is performed, showing the next instruction to be executed.

Unlike the Go command, the Trace command can be used to trace through ROM, since it works by changing the trap flag and not by modifying the code to be traced.

Examples:

T traces the execution of a single instruction.

T 3 traces the execution of the next three instructions.

T CX traces the execution as many times as indicated by the current value of the CX register. If CX is currently 100H, and the next instruction changes it to zero, the trace will still be performed 100H times.

Command: Unassemble

Syntax: U [<range>]

Description: This command is used to disassemble memory into the 8088 mnemonics and byte values.

The syntax for this command is very flexible. If you enter **U**, the disassembly starts where the last **U** command left off. The commands G, J, R, and T reset the starting point to CS:IP. If you enter **U <number>** the number is presumed to be an offset, the segment is presumed to be CS, and the length is presumed to be 20H. If you enter **U <number> <length>** the number is presumed to be an offset, and the segment is presumed to be CS.

Two sample disassemblies are shown below. Both are of the same range of memory, but the second listing was made using a symbol table. Note the difference in readability.

```
Without symbols:

063A:01AD BF2F01          MOV     DI,012F
063A:01B0 E84501          CALL    02F8
063A:01B3 8B1E2F01        MOV     BX,[012F]
063A:01B7 8EC3            MOV     ES,BX
063A:01B9 8B3E3101        MOV     DI,[0131]
063A:01BD C70687010000    MOV     WORD PTR [0187],0000
063A:01C3 C7068A010000    MOV     WORD PTR [018A],0000
063A:01C9 B94D00          MOV     CX,004D

With symbols:

              P565:
063A:01AD BF2F01          MOV     DI,012F                    ; FILE_SEGMENT
063A:01B0 E84501          CALL    P525
063A:01B3 8B1E2F01        MOV     BX,[FILE_SEGMENT]
063A:01B7 8EC3            MOV     ES,BX
063A:01B9 8B3E3101        MOV     DI,[FILE_OFFSET]
063A:01BD C70687010000    MOV     WORD PTR [SKIP_COUNT],0000
063A:01C3 C7068A010000    MOV     WORD PTR [LINE_COUNT],0000
              P565A:
063A:01C9 B94D00          MOV     CX,004D
```

Each line of the disassembly shows the address of the instruction (CS:IP) followed by the one to six bytes that make up the instruction. Next the instruction is displayed.

If an address in the instruction is an exact match with an entry in the symbol table, the symbol name is substituted for the address. For example, in the third line, address DS:012F is referenced. When the symbol table is searched, the name FILE_SEGMENT is found and displayed instead of 012F.

If you're debugging programs where DS and/or ES do not initially point to the data area(s), variable references such as this one are not shown until DS and/or ES are changed to point to the data area(s) within your program. Code references, such as the label **P565** are found, since CS must be correct for your program to execute.

If an ambiguous reference to an address is made, the symbol name is shown at the end of the disassembled instruction as a comment. This indicates that the symbol may or may not have been used in the original instruction. The most common ambiguous reference is generated by a move of an offset to a register, such as a **MOV DI,OFFSET FILE_SEGMENT**.

Examples:

U @NEW_PAGE disassembles memory starting at the symbol NEW_PAGE. The default length of 20H bytes is used.

U CS:IP L 1 disassembles memory for one instruction at the current instruction. The same result can be achieved by using **R**.

U disassembles memory starting where the last **U** command left off. If the G, J, R or T commands were used, the disassembly starts at CS:IP.

Command: Write Absolute disk sectors

Syntax: WA <address> <drive> <sectors>

Description: This command is used to write memory to absolute disk sectors.

The segment defaults to CS if no segment is specified in the address. The drive is a single digit number indicating the disk drive (0 = A, 1 = B, etc.). The sectors parameter is the starting sector number and the number of sectors to be written. The maximum number of sectors that can be written in one operation is 80H.

To use this command, DOS must not be active. See the description of the Name command for more information. This command uses DOS interrupt 26H. See the DOS manual for information on the numbering of the absolute disk sectors.

When using this command, be very careful! An absolute disk write can very easily destroy a file allocation table (FAT) or a disk directory! Usually, you will want to perform a Load Absolute, change a few bytes of memory, and the perform a Write Absolute of the data back to disk. If this is the case, be sure that the parameters used with the load and write commands are the same.

Examples:

WA DS:100 0 10 20 writes data from memory starting at DS:100 to drive A, starting at sector number 10H for 20H sectors.

WA 100 1 0 4 writes data from memory starting at CS:100 to drive B, starting at sector 0 for 4 sectors.

Command: Write File to disk

Syntax: WF [<address>]

Description: This command is used to write a file from memory to disk.

The optional address specifies where the memory image of the file begins. If the address is not specified, CS:100 is used. To use this command, DOS must not be active. See the description of the Name command for more information. Before this command can be used the Name command must be used to specify a file name.

This command can be used to write any type of file to disk. Before the file is written, be sure that BX and CX indicate the size of the file in bytes.

Examples:

WF DS:1000 writes the file defined by a Name command from memory to disk starting at DS:1000.

WF writes the file defined by a Name command from memory to disk starting at CS:100.

Command: Xlate (translate) hex number

Syntax: X <number> or XH <number>

Description: This command is used to translate a one to four digit hexadecimal number or a register to its decimal and binary equivalents.

Example:

X 1234 displays '1234h 4660d 0001 0010 0011 0100b'.

Command: Xlate (translate) address

Syntax: XA <address>

Description: This command is used to translate an address (segment and offset) into its equivalent five-byte absolute address.

The absolute address is calculated by multiplying the segment by 10H and adding the offset to the result.

Examples:

XA 1234:5 performs the above calculation and displays '12345'.

XA 1234:5678 displays '179B8'.

Command: Xlate (translate) decimal number

Syntax: XD <decimal number>

Description: This command is used to translate a one to five digit decimal number to its hexadecimal and binary equivalents.

The number must be from zero to 65,535. The number may not have any punctuation, such as commas or periods. Numbers larger than 65,535 can be translated, but the high order part is lost.

Examples:

XD 4660 displays '1234h 4660d 0001 0010 0011 0100b'.

XD 65538 displays '0002h 2d 0000 0000 0000 0010b'.

Command: Option S

Syntax: /S

Description: This command is used to make global changes to the segment numbers in the symbol table.

The entire symbol table is searched for symbols having a segment that matches the first segment entered. If a match is found, the symbol's segment is changed to the second segment entered and the symbol name is displayed. This command is used to adjust the segments of symbols when a program relocates its data areas, such as in Microsoft Pascal and FORTRAN.

Example:

/S FEED DS changes the segment of all symbol table entries that are currently FEED to the current value of DS.

/S CS CS displays the segment of all symbol table entries that match the value of CS. This is a good method for querying the symbol table without changing anything.

VI — RUN — The Program Loader

The program RUN (RUN.COM) is used to load COM or EXE files and enter the resident debugger. PERISCOPE must be installed in the protected memory for RUN to work.

RUN can also be used to load data files or no file at all. If no file is loaded, the first instruction is set to **INT 20H**, the DOS return, to prevent accidental execution of meaningless data. If a data file is loaded, be sure to use the Return to DOS option after quitting the debugger. The Continue option would execute the data file with unpredictable results.

RUN is started by entering **RUN filename.ext command-line** at the DOS prompt, where filename.ext is the file name and extension (EXE, COM or other) of the file to be loaded. The command line is the same command line used when the program is started directly from DOS. RUN adjusts the File Control Blocks (FCB's) and command line in the PSP to look like the target program had been started directly from DOS. RUN then resets the PSP address in PERISCOPE to the current PSP, so the Return to DOS option and disk I/O commands can be used.

If the file extension is COM or EXE, the disk is searched for a file of the form filename.DEF. If this file is found, it is presumed to be a record definition file. This file is then used to load record definitions into the record definition table stored in normal RAM. If the DEF file is not found, the record definition table is cleared. If an error is found in the DEF file, the record definition table will be partially loaded. See the section describing the program RS.COM for more information.

If the file extension is COM or EXE, the disk is searched for a file of the form filename.MAP. If this file is found, it is presumed to be a MAP file generated by the linker. This file is then searched for address and line references. If found, these references are added to the symbol table stored in normal RAM. If the MAP file is not found, the symbol table is cleared. If an error is found in the MAP file, the symbol table will be partially loaded. See the section describing the program TS.COM for more in-

formation.

RUN then relocates itself upward and reads the target program into memory, beginning at RUN's original location, and performs any segment relocation required by EXE files. Registers BX and CX are then set to the size in bytes of the target file. Other registers are set according to the rules for loading COM and EXE files (see the DOS manual). The 'load high' option is not currently supported.

Your program is loaded exactly where it would be if DOS were to load it under the same conditions. This feature allows RUN to be used to load memory-resident programs. Until RUN is used again, the record definition and symbol tables are preserved.

Finally, control is passed to the resident portion of PERISCOPE. Since PERISCOPE saves the state of the keyboard buffer before accepting input, do not type ahead while waiting for your program to load. Anything keyed after RUN is started but before PERISCOPE is ready for input will be read by your program or by DOS when you return to it.

When you've finished debugging your program, you can exit PERISCOPE in one of three ways — use a Go with no breakpoints set, Quit to the command menu and then Continue execution, or Quit to the command menu and Return to DOS. If you use the last option, be sure that all output files are closed and that any interrupt vectors your program has modified are reset to their original values.

VII — Technical Notes

This chapter discusses some miscellaneous technical topics, including:

- Debugging theory and limitations
- Suggested debugging techniques
- An overview of PERISCOPE's internals and data area
- The memory board

Debugging Theory

The 8088 processor provides two built-in functions that aid the debugging process. These are the breakpoint and trap flag capabilities.

The breakpoint capability uses a special single-byte code to indicate that a breakpoint is to be taken. This code causes the system to perform an Interrupt 3 when the first byte of an instruction equals CCH. This is the facility used by the Go command in PERISCOPE, for both the temporary and 'sticky' code breakpoints.

When PERISCOPE sets a code breakpoint, the original byte is saved in an internal table and the breakpoint code is inserted in its place. For this reason, it is not possible to set a code breakpoint in ROM or other unmodifiable memory.

When the breakpoint is taken, the resident portion of PERISCOPE is entered through a special entry point. This entry point indicates that a code breakpoint has occurred, signaling PERISCOPE to reverse out any code breakpoints that are currently set and then decrement the instruction pointer (IP) by one to show the correct instruction.

If an unexpected instruction contains a CCH in the first byte, PERISCOPE is unable to reset the instruction to its prior value and will disassemble the instruction as **INT 3**. You will need to manually alter the byte or skip over the instruction to continue execution of the program

being debugged.

The trap flag is the other type of 8088 breakpoint. It is set by modifying one of the flags to indicate that every instruction should be trapped. If this flag is set, the system performs an Interrupt 1 before the execution of each instruction, allowing you to single-step through a program. The trap flag is used by PERISCOPE for the Trace command and all of the non-code breakpoint commands (byte, memory, register, and word).

If an instruction outside PERISCOPE clears the trap flag, it causes any tracing currently underway to be turned off. If an external instruction sets the trap flag, PERISCOPE acts as if a Trace command had been issued.

If the button is pressed while a Trace or monitor breakpoint is in progress, it may be temporarily ignored. Press the button again if you get no response the first time.

Since the tracing logic is inside PERISCOPE, it is not possible for you to trace the execution of either Interrupt 1 or Interrupt 3.

Debugging Techniques

When you press the button to stop the execution of a program, chances are very good that you'll stop the machine in either BIOS or DOS. If you want to get back to your program, try using the Register Breakpoint. Enter **BA *** to clear any breakpoints currently set, and then enter **BR CS NE CS** to set a breakpoint when the Code Segment changes from its current value. Then enter **GT** to continue execution with the register breakpoint set. This will usually get you back to the program, or at least from BIOS to DOS or vice-versa.

To repetitively trace an instruction, use the Trace and Go commands on the same command line, separated by a semi-colon. For example, if you want to trace the execution of line 110H, enter **T;G 110** and press return. Then press F4 to repeat the trace/go combination as many times as desired.

Although PERISCOPE can be used to trace DOS, the tracing of some DOS interrupts, such as 20H, causes problems. Do not allow the **GB** or **GT** commands to trace past an **INT 20H** instruction, or a hung system will result.

To debug a memory-resident program, use RUN to load the program and its symbol table. The program will be loaded in the same location as if it were run directly from the DOS prompt. Enter **G** to install the program and return to the DOS prompt. Until RUN is used to load another program, the symbol table will remain available — ready for you at the push of the button.

If you're programming in Lattice/Microsoft C, you can get debugging information such as line numbers and address references in your MAP file by using the **-d** option with the first pass of the compiler. By defining variables as STATIC, you can cause address references to be generated for program variables (with a small increase in program size). At link time, be sure to specify a MAP file and the /L and/or /M options as desired.

If you're programming in Microsoft Pascal or FORTRAN, the addresses in the MAP file that reference data variables will be incorrect. These compilers generate false segments for DGROUP data. The actual segment used depends on the amount of memory available at run time. To correct the false segments, do the following:

- Use RUN to load the program, then execute the program until DS is modified. The best method is to go to the first line in the source program, using the symbol for the line number. The value of DS at this point is the correct segment.

- Display a known data symbol using the **D** command. The segment associated with the symbol is the invalid segment.

- Enter **/S xxxx yyyy** where xxxx is the invalid (old) segment and yyyy is the correct (new) segment. This will change all occurrences of segment xxxx in the symbol table to yyyy.

If you're calling assembly-language subroutines from a high-level language, PERISCOPE can be used to trace through the execution of the subroutine to verify that it is operating correctly. If the subroutine is linked to a compiled program, simply use **G @SUBNAME**, where SUBNAME is the name of the subroutine. If the subroutine is being called from an interpretive language such as BASIC, modify the subroutine so that the first byte contains CCH. Then when the subroutine is executed, the breakpoint (CCH) will activate the resident portion of PERISCOPE. At that point, you can modify the instruction to be a NOP

(no operation) by using **E CS:IP 90** or skip to the next instruction by modifying IP to be IP plus 1.

PERISCOPE's Internals

When the resident portion of PERISCOPE is activated via any method, the following steps are performed:

- If PERISCOPE is already active, control is returned to the calling program.
- The write-protected memory is write-enabled.
- The registers and stack are saved.
- If this is a monitor breakpoint, and the current instruction does not satisfy any of the breakpoints, control is returned to the program being debugged after reversing the above items.
- The speaker is turned off, the keyboard is turned on, and hardware interrupts are enabled.
- The temporary and 'sticky' code breakpoints are reverted as needed.
- Interrupts 1, 2, and 3 are refreshed to point to PERISCOPE.
- The state of the keyboard buffer and CRT control tables are saved.
- The current values of the BIOS interrupts used by PERISCOPE are saved and changed to their power-on values, except if the /V option was used when PERISCOPE was loaded from disk.
- Any needed screen saving/swapping is performed.
- If the button was pressed, the command list is displayed, otherwise the Register command is performed.

When PERISCOPE is exited using the Trace, Go, Continue, or return to DOS commands, the following steps are performed:

- The screen and keyboard buffers are restored to their value on entry.
- BIOS interrupts are restored to their value on entry.
- If Trace or monitor breakpoints are used, hardware interrupts are temporarily disabled.
- The registers and stack are restored to their values on entry.
- The memory is write-protected.
- If the DOS return is used, the speaker is turned off, the keyboard is turned on, and hardware interrupts are enabled.

When PERISCOPE is exited using the Short boot command, the following steps are performed:

- Interrupt 2 (NMI) is restored to its power-on value.
- If PERISCOPE was installed with the /D option, the value of Interrupt 13H is reset to its value at the time PERISCOPE was loaded from disk.
- BIOS interrupts are restored to their power-on values.
- The stack is relocated to the end of the first 64K of memory.
- The memory is write-protected.

When PERISCOPE is exited using the normal boot command, the following steps are performed:

- The stack is relocated to the end of the first 64K of memory.
- The memory is write-protected.

The requirements for the proper operation of PERISCOPE are:

- On entry, the stack must have room for 2 words, not including the three words required by the Interrupt to get from the program being debugged to PERISCOPE.

- If the trace or monitor breakpoints are in use, Interrupt 1 must point to the proper point in the resident portion of PERISCOPE.

- If the button is to be used, Interrupt 2 must point to the proper point in the resident portion of PERISCOPE. Also, programs must not disable NMI via an Out of zero to port A0H.

- If the code breakpoint is in use, Interrupt 3 must point to the proper point in the resident portion of PERISCOPE.

- The Interrupt Mask Register (IMR) located at I/O Port 21H must not be redefined after PERISCOPE is loaded from disk. This port is used to disable hardware interrupts while a single trace cycle executes. After the trace, the value read from port 21H when PERISCOPE was originally run is output to the port.

The data fields used by PERISCOPE are located at the beginning of the protected memory. The record definition PSDATA in the file SAMPLE.DEF contains the most useful of these data fields. The source file contains comments describing the various fields in the record definition. To display PERISCOPE's data area assuming the default memory address of C000:0000, enter **DR C000:0 @PSDATA**.

The Memory Board

The board uses 16K of memory and two consecutive I/O ports. The memory is configured as 8 chips of 2K by 8 static RAM, with a cycle time of 200 NS or less. The starting address of the memory is switch selectable to any 16K boundary. The starting I/O port is switch selectable to any 4 byte boundary.

The memory is write-enabled when the value DBH is output to the first of the two ports. Any other value output to this port write-protects the memory. When the button on the board or the remote button is pressed, a Non-Maskable Interrupt (NMI) is generated. The resident debugger, PERISCOPE, detects if the button was pressed by checking the high bit read from the second I/O port. If this bit is on, the button was pressed, otherwise the button was not pressed. To clear the button, output any value to the second port.

Appendix A - Error Messages

The error messages generated by programs in this package are numbered. Each program has been assigned a range of numbers for easy cross-reference. The error numbers and corresponding programs are:

1 through 39 — resident portion of PERISCOPE (PS.COM)
40 through 59 — transient or installation portion of PERISCOPE (PS.COM)
60 through 79 — RUN (RUN.COM)
80 through 89 — Symbol table size program (TS.COM)
90 through 99 — Record table size program (RS.COM)

A list of the possible error messages and an explanation of each follows:

01 - Invalid function
An unknown debugger command was entered. The first character of the command must be ?, B, C, D, E, F, G, H, I, J, K, L, M, N, O, Q, R, S, T, U, W, X, or / in upper or lower case.

02 - Invalid/missing address
An address was expected, but was not found or was found to be invalid. The address may be entered as a valid symbol, preceded by @ or a one to four-digit segment, a colon, and a one to four-digit offset. A register name may be substituted for either number.

The segment and colon may be omitted from most commands. The offset must be present for all commands requiring an address.

03 - Missing segment
Commands that modify memory (Enter, Fill, and Move) require an explicit segment. Enter the segment as a number or register, or use a symbol for the address.

04 - Invalid/missing length
The length argument was not found or was found to be invalid. If entered as **L nnnn**, the number nnnn must be greater than zero. If entered as an offset, the number must be greater than or equal to the first offset. If entered as a symbol, the symbol's segment must equal the first segment entered and the symbol's offset must be greater than or equal to the first offset.

Note that the length argument is optional for the Display and Unassemble commands. The default length for these commands is 80H and 20H bytes respectively.

05 - Unexpected input
After completion of a command, an unexpected entry was found. If multiple commands are desired, check to see that a semi-colon is placed between the commands.

06 - Missing list
No list was found for the Fill or Search commands. These two commands require a byte/string list.

07 - Missing quote
The trailing single or double quote was not found for a list.

08 - Missing operator
If the Hex arithmetic command was used, this error indicates the absence of an arithmetic operator between the two numbers. The valid operators are: +, -, *, and /.

If the memory breakpoint (BM) was used, this error indicates the absence of a read and/or write check. The valid operators are R and W in upper or lower case.

If the byte breakpoint (BB), register breakpoint (BR), or word breakpoint (BW) command was used, this error indicates the absence of a test. The valid tests are: LT, LE, EQ, NE, GE, and GT, in upper or lower case.

09 - Number is not decimal
The number entered when the translate decimal (XD) command is used must be in decimal format, with no punctuation.

10 - Invalid/missing number
A required number was not found or was found to be invalid. The number must be from one to four hex digits or a valid register name. If part of a list, or used with the Output command (O), the number must be one or two digits and a register name cannot be used.

11 - Invalid register
The register name must be AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, or IP, in upper or lower case.

12 - Invalid flag
The valid flag names are OV, NV, DN, UP, EI, DI, NG, PL, ZR, NZ, AC, NA, PE, PO, CY, and NC, in upper or lower case.

13 - Too many addresses
Too many addresses were input for the BC command (limit 16), the BB command (limit 8), the BW command (limit 8), or the G command (limit 4).

14 - Invalid sub-function
For commands using sub-functions, the sub-function must be entered immediately after the function. The Breakpoint command must be followed by an A, B, C, M, R, or W. The Display command must be followed by a space (same as the B sub-function), B, D, R, or W. The Enter command must be followed by a space or S. The Go command must be followed by a space, B, or T. The Load and Write commands must be followed by A or F. The Search command must be followed by a space or A. The Xlate command must be followed by a space (same as the H sub-function), A, D, or H.

15 - Cannot trace INT 3
An attempt was made to trace interrupt 3 using PERISCOPE. This

interrupt is off-limits, since any attempts to have the program trace itself results in total confusion.

16 - Cannot modify memory
An attempt was made to set a code breakpoint in memory that could not be modified. The memory is not present, is Read-only (ROM), or was not correctly updated with the CCH code needed for a code breakpoint.

17 - Second address less than first
For the memory breakpoint (BM) command, the second address entered is less than the first. For commands requiring a range, the second offset was found to be less than the first offset. For example, the command **D 0:100 80** is invalid, since 80 is less than 100.

18 - Unknown symbol
An unknown symbol was referenced. The symbol must be preceded by a @ and must be followed by a trailing space, carriage return, or semi-colon. The maximum symbol length is 16 characters. Lower case input is converted to upper case before the symbol or record definition table (if DR is used) is searched.

Line numbers are of the form **Xnn**, where X is the module sequence letter (A for the first module in the link map, B for the second, etc.) and nn is the decimal line number. The line number does not have leading zeroes. For example, @**A1** may be a valid symbol, but @**A01** is not.

To display the symbol names and addresses from the symbol table, use F8. To display the record definition table, use F7.

19 - Cannot use Jump
The Jump (J) command was used when the current instruction is a RET, IRET, or any form of JMP. Since control will not return to the next instruction, Jump cannot be used. Use Trace or Go instead.

20 - DOS functions not available
DOS function calls are used by the Load, Name, and Write commands. Since DOS is not re-entrant, PERISCOPE tests to be sure that DOS is not

currently active. This is done by checking a flag that is set when RUN is run and cleared when the button is pressed. The flag must be set and CS:0 must contain the bytes **CD 20** for DOS functions to be made available from PERISCOPE.

If you get this message, return to DOS, run RUN.COM, and then retry the desired command.

21 - Not enough memory
Insufficient memory is available to perform the Load command. PERISCOPE checks the memory indicated at CS:2 in the PSP to be sure enough memory is available for disk I/O.

22 - Invalid drive
One of the drive names specified in the Name command is invalid. Register AL or AH is set to FFH if the first or second file name had an invalid drive identifier, respectively.

23 - File not found
The file specified with the Name command was not found when a Load File command was used. Re-enter the Name command and try again.

24 - No room in directory
No more directory entries were found on the disk specified with the Name command when a Write File command was used. Try to write the file to another disk.

25 - Read/write error
A disk I/O error occurred when reading or writing a file or absolute sectors. Use F4 to retry the command.

40 - Number must be 1 to 4 hex digits (0-9, A-F)
All numbers associated with PERISCOPE installation options are in hex format for consistency. For the /R, /S, /T, and /V options, the number must be one or two hex digits.

41 - Not enough memory
Insufficient memory is available to install PERISCOPE. Check the amount of available memory using CHKDSK. Boot the system or reduce the space PERISCOPE requires in RAM by adjusting the /R, /S, or /T installation options.

42 - Invalid command line input
An unexpected entry was found on the PERISCOPE command line. The valid entries are: /A, /D, /H, /M:nnnn, /P:nnnn, /R:nn, /S:nn, /T:nn, and /V:nn, where nn represents a one or two-digit hex number and nnnn represents a one to four-digit hex number. Entries may be separated by spaces.

43 - Interrupt must be 09H, 10H, 16H, 17H, or 1CH
The /V option specified an interrupt number other than the ones listed above.

44 - Unable to modify protected memory
PERISCOPE was not able to install itself in the protected memory. Check the port setting on the board and the port number specified with the /P option, if any.

45 - Unable to protect memory
After protecting the memory on the board, PERISCOPE was able to modify the supposedly protected memory. Check the memory setting on the board and the memory address specified with the /M option, if any.

46 - Copy of program in protected memory is invalid
The copy of PERISCOPE in the protected memory does not agree with the temporary copy in RAM. Check that the memory board is properly seated in the expansion slot and that the memory chips on the board are properly seated in their sockets.

47 - Screen size must be from 4 to 20H (32) KB
The size of the original screen specified with the /S option must be from 4 to 20H K. Note that the number is in hex!

48 - Symbol table size must be from 1 to 3FH (63) KB
The size of the symbol table specified with the /T option must be from 1 to 3FH K. Note that the number is in hex!

49 - Too much memory allocated to screens and tables
If a version of DOS prior to 2.00 is used, the total size of the original screen (4 to 32K), the debug screen (always 4K), the record definition table (1 to 32K), the symbol table (1 to 63K), and the help file (if used) must be 63K or less. For DOS 2.00 or later, only the individual limits and the total memory available apply.

50 - Record table size must be from 1 to 20H (32) KB
The size of the record definition table specified with the /R option must be from 1 to 20H K. Note that the number is in hex!

51 - HELP file not found
The help file, PSHELP.TXT was not found on the default disk drive. Copy the help file to the default disk or omit the /H option.

52 - Unable to read HELP file
An error occurred reading PSHELP.TXT. Restore the file from your backup disk.

60 - File not found
RUN was not able to find the specified file. Check the file name and restart RUN.

61 - EXE Header not found
A file with an extension of EXE was specified, but the header record identifying the file as a valid EXE file was not found. Regenerate the EXE file and restart RUN.

62 - Unable to read xxxxxxxxxxx
An error occurred reading the specified file. The file name and extension are shown in place of the x's. Check to be sure the drive is ready and that

the file size shown by DIR indicates the true file size.

63 - Not enough memory
Insufficient memory is available for RUN to load the desired program. Check the amount of available memory using CHKDSK and re-boot as needed.

64 - PERISCOPE not installed
RUN cannot run without PERISCOPE installed in the protected memory. Install PERISCOPE and restart RUN.

65 - PERISCOPE not installed correctly
RUN was unable to set the Program Segment Prefix in the protected memory. Reload PERISCOPE and try again.

66 - Unable to load MAP file
A MAP file was found for a COM or EXE file, but RUN was unable to load it correctly. Check the format of the MAP file and the size required for the MAP file using TS.COM. The MAP file must be in the format as produced by LINK — some editors may cause subtle reformatting of the file.

If the space required by the MAP file is greater than the symbol table size, as much of the MAP file is loaded into the symbol table as possible. Use F8 to see the symbols that were loaded.

67 - Unable to load DEF file
A DEF file was found for a COM or EXE file, but RUN was unable to load it correctly. Check the format of the DEF file and the size required for the DEF file using RS.COM.

If the space required by the DEF file is greater than the record table size, as much of the DEF file is loaded into the record table as possible. Use F7 to see the records that were loaded. Note that the last record will usually be only partially defined.

80 - MAP file not found
The table size program was not able to find a file of the specified name with an extension of MAP.

81 - Unable to read MAP file
An error occurred reading the MAP file. Regenerate the file and try again.

82 - Line xxxxx of MAP file is not in correct format
The MAP file is not in the format expected. The line number indicates the line in the MAP file where the error occurred.

If you've used a text editor to modify the MAP file, be sure to save it in its original format — with no embedded tab characters or high bits set. The format as produced by the linker may have changed — try using another version of LINK. If this is the problem, please advise us of the situation as soon as possible.

83 - Not enough memory
Insufficient memory is available for TS.COM to load the MAP file. Check the amount of available memory using CHKDSK and re-boot as needed.

90 - DEF file not found
The record size program was not able to find a file of the specified name with an extension of DEF.

91 - Unable to read DEF file
An error occurred reading the DEF file. Check the file's format and try again.

92 - Line xxxxx of DEF file is not in correct format
The DEF file is not in the format expected. The line number indicates the line in the DEF file where the error occurred. Check the format of the DEF file as defined in the section describing the program RS.COM.

93 - Not enough memory
Insufficient memory is available for RS.COM to load the DEF file. Check the amount of available memory using CHKDSK and re-boot as needed.

Index

A

address parameter 5-11
address, effective 5-46
arithmetic, hex 5-36
arithmetic operator parameter 5-11, 5-36
ASM programs 5-7, 7-3

B

BASIC programs 5-9, 7-3
BIOS 1-5, 5-2, 5-4
board, memory (see SUBMARINE)
boot option 1-2, 5-8, 5-45, 7-5
boot option, short 1-2, 5-2, 5-9, 5-45, 7-5
breakpoints 1-3, 5-15 to 5-21, 5-32 to 5-35, 5-38, 7-1, 7-2
breakpoint clear command 5-16
breakpoint display command 4-3, 4-4, 5-15
breakpoint on byte command 5-17, 5-34
breakpoint on code command 4-3, 5-18, 5-32
breakpoint on memory command 5-19, 5-34
breakpoint on register command 5-20, 5-34
breakpoint on word command 4-3, 4-4, 5-21, 5-34
buffers 5-5
byte parameter 5-11

C

C programs 5-7, 7-3
clear screen command 4-4, 5-39
color-graphics adapter 5-4
COM files 4-2, 6-1, 6-2
command list 5-8
commands 5-14 to 5-58
compare command 4-5, 5-22
conflicts, memory 3-1
conflicts, ports 3-1
continue option 5-8, 5-45, 7-4
copy (see move command)
Ctrl-Break key 5-10
Ctrl-PrtSc key 5-10
Ctrl-S key 5-10

D

debug option 5-8, 5-45
debugging techniques 7-2, 7-3
debugging theory 7-1, 7-2
decimal number parameter 5-11
decimal number, translate command 5-57
DEF file 4-1, 5-6, 5-26, 6-1
Del key 5-9
DIP switches 1-1, 3-2 to 3-5
disassemble (see unassemble)
disk drive parameter 5-11
display byte command 4-2, 4-4, 4-5, 5-23, 5-24
display double word command 4-4, 5-25
display record command 4-2, 5-26, 5-27

display word command 1-4, 4-4, 5-28
DOS 1-5, 5-5, 5-40, 5-41, 5-43, 5-54, 5-55, 7-2
DOS, return to option 1-2, 5-8, 5-45, 7-4
drive parameter 5-11

E

effective address 5-46
enter command 4-5, 5-29
enter symbol command 5-30
errors A-1 to A-10
Esc key 5-10
EXE files 5-41, 6-1, 6-2

F

F1 key 5-9
F3 key 5-9
F4 key 5-9, 5-32
F7 key 4-2, 5-10
F8 key 4-3, 5-10
F10 key 5-10
FCB 4-1, 5-26, 5-27, 5-43, 6-1
File Control Block (see FCB)
fill command 4-5, 5-31
flag 5-11, 5-47, 5-48
FORTRAN programs 5-7, 5-58, 7-3
function parameter 5-12

G

go command 1-4, 4-4, 5-32, 5-33, 7-4
go using breakpoint command 4-3, 5-34
go using trace command 5-35
graphics 5-4

H

help 2-1, 4-1, 5-2, 5-3
help command 4-2, 5-14
hex arithmetic command 4-5, 5-36

I

IMR 7-5
input command 5-37
Ins key 5-9
interrupts 5-2, 5-4

J

jump command 4-4, 5-38

K

keyboard usage 5-9, 5-10
klear command 4-4, 5-39

L

length parameter 5-12
load absolute sectors command 5-40
load file command 5-41
line number, program 1-4, 5-7
linker 5-7
list parameter 5-12

M

MAP file 4-1, 5-7, 6-1
memory board (see SUBMARINE)
memory, protected 3-1, 3-4, 3-5, 5-3, 7-6
monitor, alternate 5-2
move command 4-5, 5-42
multiple commands 1-5, 5-10, 5-29

N

name command 5-43
name parameter 5-12
NMI 1-1, 7-5, 7-6
non-maskable interrupt (see NMI)
number parameter 5-12
number, translate command 5-56, 5-57

O

offset parameter 5-12
options, command list 5-8
output command 5-44

P

parameters 5-11
parity error 1-2, 1-5, 5-8
Pascal programs 5-7, 5-58, 7-3
PERISCOPE 1-1 to 1-5, 2-1, 5-1, 6-2
  buffers 5-5
  command line 5-2 to 5-5
  command list 5-8
  default values 5-1, 5-2
  installation 4-1, 5-1 to 5-5
  internals 7-4 to 7-6
port 5-37, 5-44
port parameter 5-12
ports, memory protect 3-1 to 3-3, 5-3, 7-6
Program Segment Prefix (see PSP)
PSP 4-1, 4-2, 5-8, 5-12, 5-26, 5-27, 5-43, 6-1
PS.COM (see PERISCOPE)
PSHELP.TXT (see help)

Q

quit command 4-5, 5-45

R

range parameter 5-12
record definitions 4-2, 5-3, 5-6, 5-10, 5-13, 5-26, 5-27, 6-1
record size (see RS.COM)
register parameter 1-4, 5-12
register command 4-4, 5-46, 5-48
requirements, system 1-6
RS.COM 2-1, 5-3, 5-6
RUN.COM 1-2, 2-1, 4-2, 5-5, 5-8, 6-1, 6-2

S

sample program 2-2, 4-1, 4-2
screen 1-4, 5-2, 5-51
search address command 4-3, 5-50
search command 4-5, 5-49

sectors parameter 5-13
segment parameter 5-13
semi-colon key 5-10, 5-29
Shift-PrtSc key 5-10
string parameter 5-13
sub-function parameter 5-13
SUBMARINE 1-1, 3-1 to 3-6, 7-6
  conflicts 3-1
  installation 3-5, 3-6
  switch settings 3-2 to 3-5
  technical notes 7-6
SW1 3-2, 3-3
SW2 3-2, 3-4, 3-5
switch, DIP (see DIP switch)
switch, push button 1-1, 1-2, 3-6, 7-6
switch, remote 1-2, 3-6, 7-6
symbol 1-4, 4-2, 4-3, 5-10 to 5-13, 5-46, 5-47, 5-52, 5-53
symbol parameter 5-13
symbol table 5-4, 5-7, 5-30, 5-58, 6-1, 7-3
symbol table size (see TS.COM)
system requirements 1-6

T

table size, symbol (see TS.COM)
test parameter 5-13
trace command 1-4, 4-2, 4-4, 5-51, 7-4
translate address command 5-56
translate decimal number command 5-57
translate number command 4-4, 5-56
TS.COM 2-2, 5-4, 5-7
tutorial 4-1 to 4-5

U

unassemble command 4-2, 4-3, 4-4, 5-52

W

write absolute sectors command 5-54
write file command 5-55
write-protected memory 1-1

X

xlate (see translate)

PERISCOPE Version 1.10 Addendum

This addendum describes the changes and additions to PERISCOPE. The entire manual and quick-reference card will be reprinted when the next version of PERISCOPE is released. The changes and additions include:

- User-specified windows for data, stack, register, and disassembly information. A pause option has been added to keep unwindowed information from scrolling off the screen.
- Faster output to a color monitor, especially when a no-flicker color card is used. You can also specify the color of PERISCOPE's screen.
- The Shift-PrtSc key combination can be used to activate PERISCOPE.
- Faster symbol loading and support for program symbols when Phoenix's PLINK and Digital Research's LINK86 linkers are used.
- Disassembly changes and 80286 support.
- Ability to debug device drivers and non-DOS programs.
- Other miscellaneous items, including new & changed error messages.

The changes are described in detail below.

Windows

PERISCOPE can window data, stack, register, and disassembly information, based on a command-line parameter entered when PS.COM is run. Once windows are established, the windowed data is displayed at a constant location on the screen and is updated after each command.

The format of the command-line parameter is **/W D:nn R S:nn U:nn**, where **/W** indicates that windowing information follows. The tokens D, R, S, and U indicate the type of data to be windowed. The tokens are optional and may be in any order. If a token is omitted, the corresponding type of information will not be windowed. The windows are displayed in the same order as the tokens are encountered on the command line.

The **D** window shows data in byte (db), word (dw), or double word (dd) format. The address used initially defaults to 0:0, until a register or display command is used. The window continues to show the same address until another register or display command is used. The output of the display record (dr) command is not shown in this window. Duplicate lines are not suppressed for windowed data.

The **R** window shows register and flag information. The length is fixed at two lines.

The **S** window shows the current stack. The display is the equivalent of **DW SS:SP**.

The **U** window shows the effective address of any memory reads or writes performed by the current instruction when the first instruction displayed starts at CS:IP. The address used initially for the disassembly defaults to CS:IP and is reset to CS:IP each time a G, J, R, or T command is used. Any area of memory can be disassembled by using the U command with the desired address. If you want to page through memory, enter the U command with no address.

The **:nn** defines the length of the window (in hex as with all other command-line parameters). If no length is specified, a default will be used. The maximum length for any one window is 10H (16) lines and the total area that can be windowed is 21 lines, including a separator line following each window. When a length specification is used, at least one space must follow the number.

The default, minimum, and maximum lines for each of the four window types are:

	Default	Minimum	Maximum
Data	4	1	10H
Register	2	2	2H
Stack	2	1	10H
Unasm	4	4	10H

Examples

/W D:8 R — Window data in the first 8 lines of the screen, followed by two lines of register information. A total of 12 lines are used for windows, including the two separator lines.

/W SRU — Window the stack in the first two lines of the screen, followed by two lines of register information, followed by four lines of disassembly. A total of 11 lines are used for windows.

Pause

A pause option has been added to PERISCOPE. To use it, press F6 when the cursor is at the beginning of a command line. If pause is off, it is turned on and the message **Pause on** is displayed. If pause is on, it is turned off and the message **Pause off** is displayed.

When pause is on, the message **Press any key** is displayed each time a single command fills the un-windowed area of the screen. This keeps the display from scrolling away too quickly, especially when most of the screen is being used for windows.

Color Monitors

If you use a color monitor for PERISCOPE's display, two new features are available for you — faster output and color selection.

The faster output is available if you have a no-flicker color card. The original IBM color card suffers from 'snow' when you attempt to write directly to the screen buffer. A flicker-free card allows you to write directly to the screen without any snow. If you have one of these cards, specify the **/F** parameter on the command line when starting PERISCOPE (PS.COM).

You can now have your debugger screen in living color if you have a color monitor. To set the color, enter **/C:nn** on the command line when starting PERISCOPE. NN is the hex (as always!) value of the color attribute to be used. This number may be from 1 to FFH. The border of the screen is set to the same color as the background.

To calculate the number you want (I use 17H — gray on blue) see your machine's technical reference manual or the chart below.

The color attribute byte is a nice bit of bit packing. The layout of the bits is shown below.

```
Bit number      7    6    5    4    3    2    1    0

Use             X    R    G    B    H    R    G    B
                     ----------        ----------
                     background        foreground
                       color             color

X - blink if 1, else no blink
R - red gun on if 1, else off
G - green gun on if 1, else off
B - blue gun on if 1, else off
H - high-intensity if 1, else normal intensity

The RGB combinations are:
Green plus Blue is Cyan
Red plus Blue is Magenta
Red plus Green is Brown (Yellow if high intensity)
Red plus Green plus Blue is Gray (White if high intensity)
```

Assuming that you wanted high intensity red on green (holiday colors?), you'd use 0010 1100 binary, or **/C:2C**. Similarly, if you want cyan on black, use 0000 0011 binary, or **/C:3**. There are some illegal color combinations that PERISCOPE won't allow. These include 0, 8, 80H, and 88H which are all variants of black on black, and other similar situations where the foreground and background colors are the same, such as 77H and F7H.

Shift-PrtSc To Activate PERISCOPE

A **/K** command-line option has been added to enable the Shift-PrtSc keys as an entry to PERISCOPE. Specify **/K** on the command line when PS.COM is run. Shift-PrtSc will not work when interrupts are disabled — try using the break-out switch in this case.

Note that the CS:IP shown after using Shift-PrtSc to enter PERISCOPE is always the same — the instruction after INT 5 in the keyboard handler (INT 9). Once the **/K** option is used, the use of Shift-PrtSc will activate PERISCOPE, until the system is re-booted using a normal boot, since DOS does not modify INT 5.

The same effect as the **/K** command can be achieved by using PERISCOPE's move command — **M 0:8 L4 0:14**. If you're debugging software that uses the BOUND instruction, you should use /K so an exception won't cause your printer to print forever!

Symbols

Large .MAP files are relatively slow to load, since RUN.COM must extract the symbols each time the corresponding program is executed. A new option has been added to the TS.COM program that improves the loading of large symbol tables. Enter **TS progname/S** to analyze the MAP file and create a file of the form progname.PSS that is a memory image of the symbol table for the program.

When RUN is executed, it first looks for the .PSS file. If the PSS file is found, it is used for the symbol table. If no PSS file is found, the MAP file is used. Be careful — if you have created a PSS file and then re-link the program without creating a new PSS file, the old PSS file will be used. It's a good idea to create the new PSS file from the batch file used to compile and link your program.

If the PSS file is too big to fit in the space allocated by PERISCOPE, no symbols will be loaded. This is in contrast to the MAP file, where the symbol table may be partially loaded before an error occurs.

PERISCOPE now supports Digital Research's LINK86 (version 1.3) and Phoenix's PLINK (version 1.3). Since RUN knows nothing about the formats used by these two linkers, TS.COM must be used to generate a PSS file for both of them. Line numbers are not currently available as symbols from either of these linkers.

For LINK86, enter **TS progname/D/S**. The **/D** option tells TS that the input file was generated by DRI's linker. The presumed input file extension for this option is SYM, not MAP! The **/S** option tells TS to write the PSS file to disk, for later use by RUN. At link time, be sure to specify a SYM file.

For PLINK, enter **TS progname/P/S**. The **/P** option tells TS that the input file was generated by Phoenix's linker. The presumed input file extension for this option is MAP. The **/S** option tells TS to write the PSS file to disk, for later use by RUN. At link time, be sure to specify a MAP file and the **G** report (all symbol information is read from the **G** report).

Disassembly And 80286 Support

Since it is common practice to move and compare ASCII bytes, PERISCOPE's disassembly now shows the ASCII equivalent of immediate bytes referenced by various instructions. If the data byte is from 20H to 7FH, the ASCII equivalent is shown at the end of the line as a comment, in quotes. For example, the instruction CMP AL,'B' is shown as:

```
XXXX:XXXX 3C2C          CMP     AL,42                   ; 'B'
```

The comment is shown on various byte instructions, including: ADC, ADD, AND, CMP, MOV, OR, SBB, SUB, TEST, and XOR.

Illegal instructions are shown as **???** rather than possibly showing **DB nn**, where nn varies.

For the IBM AT, PERISCOPE may be used in real (8086) mode only. The exception interrupts 6 (invalid opcode) and 0DH (segment overrun) are intercepted by PERISCOPE, with CS:IP pointing to the offending instruction.

PERISCOPE disassembles the real mode instructions of the 80286. Protected mode instructions are not yet supported. The new supported instructions are:

```
PUSH immediate value                PUSHA
POPA                                IMUL immediate value
Shift/rotate by immediate value     INSB and INSW
OUTSB and OUTSW                     ENTER
LEAVE                               BOUND
```

The effective address calculations and displays for the above instructions are also supported, with the exception that the stack shown as affected by the ENTER instruction is limited to a single PUSH BP and does not include the PUSH that is done for each nesting level.

A minor change has been made to the effective address display to allow for the long display required by instructions such as PUSHA and POPA. Instead of displaying the reads and writes at fixed locations within the line, the read (if any) is displayed, followed by the write (if any).

Debugging Device Drivers And Non-DOS Programs

The short boot option on PERISCOPE's command list performs an INT 19H, and leaves NMI (INT 2) intact. If you are debugging non-DOS or pre-DOS programs, you can use the break-out switch after a short boot to get back into PERISCOPE. If the timing is critical, as in the situation where you need to debug device driver initialization code, embed an INT 2 in the code itself.

INT 5 will also remain unchanged across a short boot. If you've used the /K option to allow entry to PERISCOPE via Shift-PrtSc, these keys will still work after the short boot, presuming no change by another program.

PERISCOPE normally uses RAM external to the Submarine board for screen buffers, record tables, symbol tables, and the on-line help file. If you need to keep PERISCOPE from using any memory other than what is on the Submarine board, you may use the command-line parameters **/S:0 /R:0 /T:0**. This sets the first three buffers to a length of zero, instead of their defaults of four, one, and one KB respectively. Be sure NOT to use the **/H** option! A screen buffer size of zero is also achieved by using the **/A** parameter, indicating that both monochrome and color displays are available. This method is much preferred over using **/S:0**, since the latter does not preserve the original program's screen.

In many cases when you are debugging device drivers or non-DOS programs it will be sufficient to place the external tables somewhere above the first 64K of memory in the system, instead of specifying zero length. To this end, PERISCOPE displays a message that indicates where the tables start and their total length in paragraphs. If nothing else, you can rerun PS.COM until the starting address is greater than 1000:0000.

Miscellaneous

Each of the .COM programs in the PERISCOPE system (PS, RS, RUN, and TS) display help information when the program name is followed by a question mark. For example, to get help on PERISCOPE's command-line parameters, enter **PS ?**.

The following products conflict with Submarine's default port setting:

- Tecmar fixed disks documented as using port 700H (really 300H)
- 3Com Ethernet card (ports 300H through 30FH)

If you use either of the above items, be sure to change the port settings as described in Chapter III.

IBM's Enhanced Graphics Adapter conflicts with Submarine's default memory setting — the EGA's ROM is addressed at C000:0000. If you have an EGA, change Submarine's memory address to C400:0000 or some other value.

If an 8087 is used with interrupts enabled, an error will cause a NMI. Since PERISCOPE uses the NMI, the command list will be displayed. Use the **C** option to continue, or use the **D** option to enter the debugger. Since the 8087 may interrupt the 8088 at any point, CS:IP may contain any value.

The following are corrections to the manual.

(p. 1-1, par. 4) The minimum RAM workspace external to PERISCOPE is zero K.

(p. 1-6) The PERISCOPE system also works on the IBM AT, Compaq, Columbia, and Leading Edge computers. When using a Leading Edge computer, PERISCOPE must be patched to work correctly. The memory board may be used in an IBM Expansion Chassis if desired.

(p. 2-1, par. 3) The minimum size of the external data areas is zero K.

(pp. 2-2 and 5-1) TS.COM is also used to generate .PSS symbol files using any of the supported linkers.

(p. 3-6, last par. and p. 5-8 first par.) When the break-out switch is pressed, PERISCOPE goes directly to the debugger, bypassing the command list. The command list is displayed when a Quit command is used to exit the debugger.

If the break-out switch doesn't work when PERISCOPE is first installed, check the DIP switch setting for the 8087. If you have an 8087 installed, the switch should be OFF. If you don't have an 8087, the switch should be ON.

(p. 5-2) Add the following to PERISCOPE's defaults:

- Interrupt vector 8 is saved and restored by PERISCOPE
- Windows are presumed off
- The screen color is set to 7 (gray on black)
- Slow color output is used

(pp. 5-2 thru 5-5) The new command-line options are ?, /C, /F, /K, and /W as described above. The minimum size for the /R, /S, and /T options is zero. Interrupt 8 is saved/restored by PERISCOPE, unless overridden by the **/V:8** command-line option.

(p. 5-4, par. 1) If the /A option is used, the /S option, if any, is ignored and no memory is reserved for the original or debug screen buffer. The /S option is only required for single-monitor systems where a 4K screen buffer is too small.

(p. 5-5, par. 2) The example should be broken into two examples —

PS /A — Use two monitors; one for the program being debugged and one for the debugger.

PS /S:10 — Reserve 16K to save the original (graphics) screen on a single-monitor system.

(p. 5-7) See the discussion of symbols, above. The module-naming convention for source code line numbers has been extended. The first 26 modules are referred to as A through Z. Subsequent modules are referred to as AA through AZ, BA through BZ, etc.

(p. 5-8, par. 3) PERISCOPE does not beep, to prevent the modification of the sound generated by the interrupted program.

(p. 5-10) Function key F6 toggles the pause state as described above. Function key F9 performs the same function as Ctrl-S — a single-key method of suspending output.

(p. 5-34) The instruction following an INT instruction is not 'seen' by PERISCOPE, since the trap flag is not yet turned back on.

(p. 5-38) If the current instruction is any form of RET, IRET, or JMP, PERISCOPE generates a Trace command to get to the next instruction, which is not necessarily at the same level.

(p. 6-1) If a .PSS file is found, it is used instead of the .MAP file.

(p. 7-2, par. 2) If an external instruction sets the trap flag, PERISCOPE ignores it.

(p. 7-3, par. 2) By defining data variables outside the MAIN, you can cause address references to be generated in the MAP file.

(p. 7-4, par. 2) The command list is also displayed if exception interrupts 6 or DH occur on the IBM AT.

(p. 7-4, par. 3) Hardware interrupts are not disabled during a trace or monitor operation.

(p. 7-5, par. 1) Int 2 is left pointing to PERISCOPE.

(p. 7-5, last par.) The IMR is not used to disable interrupts.

Error Messages

The new and changed error messages are:

19 - Table full or invalid
The record or symbol table was found to have a logical error or is completely full. Try using an undefined record or symbol in a display statement. If the error occurs, the table has a logical error, otherwise the table is full.

If the table is invalid, chances are good that it was destroyed by a runaway program. If you have done a short boot and have not re-installed PERISCOPE, be aware that the memory previously reserved by DOS is no longer reserved.

43 - Interrupt must be 08H, 09H, 10H, 16H, 17H, or 1CH
The /V option specified an interrupt number other than the ones listed above.

47 - Screen size must be from 0 to 20H (32) KB
The size of the original screen specified with the /S option must be from zero to 20H K. Note that the number is in hex!

48 - Symbol table size must be from 0 to 3FH (63) KB
The size of the symbol table specified with the /T option must be from zero to 3FH K. Note that the number is in hex!

50 - Record table size must be from 0 to 20H (32) KB
The size of the record definition table specified with the /R option must be from zero to 20H K. Note that the number is in hex!

53 - Port number must be from 100H to 3FCH
The port number specified with the /P option must be from 100H to 3FCH. Note that the number is in hex!

54 - Memory specification conflicts with memory used by DOS
The memory address specified with the /M option conflicts with DOS memory. Use a higher address, outside the range of DOS memory.

55 - Color attribute must be from 01H to FFH and foreground color must not equal background color.
The number specified with the /C option indicates a color combination that will display nothing, i.e. the foreground and background colors are the same. Choose another color and remember that the number is in hex!

56 - Incorrect window specification
The parameters specified with the /W option were found to be in error. The window specification may contain the tokens D, R, S, and U in any order, in upper or lower case. If a number is entered, it must be of the form **X:nn**, where X is the token, and nn is from 1 to 10H. For the **R** token, the number is ignored and presumed to be two. A number must be followed by a space, a slash indicating the start of another option specification, or a carriage return. The total number of windowed lines, including a separator line for each window, must be 21 or less.

64 - PERISCOPE (Version x.xx) not installed
RUN cannot run without the corresponding version of PERISCOPE installed in the protected memory. Install PERISCOPE and restart RUN.

68 - Unable to load PSS file
An error occurred when RUN attempted to load the .PSS file into the symbol table. Usually the .PSS file is larger than the space reserved for the symbol table when PERISCOPE was started. If this is the case, restart PERISCOPE with a larger symbol table, otherwise check the file and try again.

69 - Logical error in symbol table
A logical error was found in the symbol table during final processing. If a .PSS file is used, regenerate it and try again. If a .MAP file is used, please report the error immediately.

84 - Unable to write PSS file
A disk error occurred when TS attempted to write the .PSS file. Check the disk and try again.