Periscope

The Undercover Debugger
For The IBM Personal Computer

by Brett Salter

Periscope Manual Version 2.01

Published by Data Base Decisions • 14 Bonnie Lane
Atlanta, GA 30328 • USA • Telephone: 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 this product.

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 items included in this package 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

Preface

Obtaining Updates

Be sure to complete and return the enclosed registration card! The first update released after you purchase Periscope is sent to you free-of-charge if you're a registered user. Registered users also receive notification of future updates. There is a nominal charge for these subsequent updates.

The Two Models Of Periscope

This manual describes two different models of Periscope. Periscope I includes the Submarine memory board with 16K of write-protected RAM and a remote break-out switch. Periscope II does not include the memory board, but does include a remote break-out switch. The two models of Periscope are functionally very similar, but there are some operational differences. We use an arrow ➧ to point out the differences in this manual.

➧ The version number (as shown on the diskette label and by the programs PS.COM and RUN.COM) has an 'S' suffix for Periscope II. Do not try to use the Periscope I software without the Submarine board or use the Periscope II software with the Submarine board.

Learning Periscope

Ideally you should read the entire manual and take the tutorial in Chapter IV before you begin using Periscope. While you may not remember everything you read, you will get a good idea of the scope of Periscope and you'll learn to use it faster.

However, there's something to be said for learning by doing, so if you're anxious to get started, here are the things you absolutely must do:

Step 1 -- Back up the Periscope disk (Chapter II)
Step 2 -- Install the Equipment (Chapter III)
Step 3 -- Take the tutorial (Chapter IV) or Install the software (Chapter V)
Step 4 -- Load your program (Chapter VII)

Refer to the chapters in parentheses for details on each step. We strongly recommend that you take the tutorial in Step 3. You'll install the debugger software during the tutorial, and you'll get a feel

for Periscope's overall capabilities that you won't get by merely installing the software.

Using The Manual

Here is how the manual is organized:

Chapter I -- Introduction -- gives you an overview of Periscope, its features, benefits, and requirements.

Chapter II -- Backing up the Disk -- gives you a step-by-step procedure for backing up the Periscope disk and describes each file on the disk.

Chapter III -- Equipment Installation -- this chapter describes how to check your configuration for conflicts with the Submarine board's default settings, how to change the board's settings if you have conflicts, and how to install the board and break-out switch.

➧ For Periscope II, this chapter describes how to install the break-out switch.

Chapter IV -- Tutorial: Up Periscope -- walks you through a session using Periscope to 'debug' a simple assembler program.

Chapter V -- Installing the Periscope Software -- describes how to install the debugger software; explains the installation options and the alternate methods of installing Periscope.

Chapter VI -- Debugging with Periscope -- defines the keys, the quit options, the command parameters, and the debugging commands.

Chapter VII -- RUNning Your Program -- describes how the program loader, RUN.COM, works.

Chapter VIII -- Using The Periscope Utilities -- describes how to use the various utility programs included in the package.

Chapter IX -- Technical Notes -- discusses debugging theory, CPU differences, DOS, debugging techniques, debugging device drivers and non-DOS programs, Periscope internals, the Submarine board, and the IBM Enhanced Graphics Adapter (EGA).

Appendix A -- Error Messages -- explains Periscope's error messages.

I -- Introduction

Welcome!

Thank you for choosing Periscope as your debugging system. You've made your life as a software developer easier, because Periscope gives you the tools you need to find and fix your programs' bugs quickly! The sooner you begin using Periscope, the sooner you'll start saving yourself many hours of debugging time. While you're getting acquainted with Periscope, remember that we support what we sell. If you run into a problem or have a question, just give us a call.

The following topics are covered in this chapter:

- Items Included In The Periscope Package
- What Makes Periscope Different
- Features And Benefits
- Symbols -- Your Road Map
- System Requirements

Items Included In The Periscope Package

The Periscope Debugging System includes:

1) The Submarine board, with 16K of write-protected RAM

The board's purpose is to protect the debugger code. It is easily installed in one of your computer's expansion slots, and the memory and ports it uses are switch-selectable (see Chapter III). During the software installation procedure (Chapter V), the critical portion of the Periscope software is loaded into the board's memory, and the memory is write-protected. This means that you don't have to worry about a runaway program corrupting Periscope.

➧ The Submarine board is not included in Periscope II -- ignore all references to the memory board if you're using Periscope II.

2) A remote break-out switch

The switch located on the board's mounting bracket enables you to stop an executing program and enter Periscope at any time. The optional remote switch that plugs into the phono jack on the board's mounting bracket (see Chapter III) performs exactly the same function. This break-out capability means that you can interrupt the system any time, to debug

the executing program or just to see what's going on. It is especially valuable when your system is hung.

➧ For Periscope II, the break-out switch taps in to an expansion slot that is already in use, so no additional slots are required. See Chapter III for installation instructions.

3) A diskette, with the Periscope debugger software and related files

The diskette contains the debugger software, plus a sample program and files used in the tutorial (Chapter IV); Periscope utility programs (Chapter VIII); and a Periscope demo program. (See Chapter II for a description of each file on the diskette).

The critical portion of the debugger code is loaded into the board's protected memory during software installation. The remaining non-critical code and data areas are loaded into normal RAM. You can control the amount of normal memory to be used by Periscope -- from none to over 128K -- but Periscope's capabilities are severely limited when you choose to operate with only the board's internal memory. (Check the type of the Periscope commands in Chapter VI to determine which commands require external memory.)

➧ For Periscope II, all of the debugger code and data areas are loaded into normal RAM.

4) A quick-reference card

The quick-reference card provides you with a concise list of Periscope's commands, command parameters, installation options, and key usage, all of which are described in detail in Chapters V and VI.

5) This manual

This manual's purpose is to provide you with the training you need to learn Periscope (Chapter IV) and with the technical information you need to become an expert Periscope user (Chapters V through IX). We strongly urge you to read the entire manual!

What Makes Periscope Different

In its review of Periscope, the Boston Computer Society says, "Periscope is a great debugger with something unique to offer systems-level programmers

...". Periscope's differences are in fact valuable to ALL programmers! It is designed to be tough and dependable, to give you control when you need it, regardless of the type of programming you do. It's fast and easy to use, too. As the reviewer concludes, "We were impressed by Periscope's very fast response in all its operations. It is a pleasure to use, and a refreshingly different product ... offers great value and unique advantages."

One of Periscope's key differences is its thorough crash recovery capability. The protected memory keeps runaway programs from interfering with the operation of critical debugger code. The break-out switch lets you check out problems when they occur, without having to stop and load your program under debugger control. Periscope is highly dependable because it saves the state of the machine when it's activated, then restores this saved state when done. Even when hardware interrupts are disabled, you can recover the machine by pressing the break-out switch.

➧ Periscope II does not offer the same level of crash recovery as Periscope I since it does not include the write-protected Submarine board.

You can use Periscope to debug device drivers, non-DOS and memory-resident programs. You can even debug DOS because Periscope uses BIOS rather than DOS function calls for most of its operations.

Two of Periscope's unusual features are the ability to stop on reads and writes to ranges of memory and the ability to set up record definitions or templates for displaying memory. C programmers find the breakpoint capability invaluable for resolving pointer problems. Memory templates are valuable regardless of the language you use. They allow you to display memory in a meaningful format, making the debugging process faster and easier. A sample record definition for the PSP is included in the Record Definition File on the diskette -- when you use it, the PSP looks like a PSP with all its 'fields' appropriately labelled.

Features And Benefits

Periscope's job is to make software development quicker and easier for programmers. That's why it's designed to be easy to learn and use, reliable, comprehensive, and fast. The key features below are grouped according to these design criteria.

Periscope is EASY TO LEARN AND USE because it gives you:
- Optional on-line help
- Commands similar to Debug's
- Optional windows that you can define and re-define to display disassembly, stack, register, and data information
- Multiple memory display formats, including ASCII, byte, integer, signed integer, word, and double word
- The ability to define your own memory display templates and to display memory using these definitions while debugging
- Symbols, source line numbers, and source code instead of addresses (See the discussion on symbols later in this chapter)
- The ability to assign command sequences to function keys
- The ability to switch from your program's screen to Periscope's screen and back with a single keystroke
- The ability to use a two-monitor system -- Periscope uses the monitor not used by your program
- The ability to display available symbols and interactively add and change symbols

Periscope is RELIABLE because:
- It gives you dependable crash recovery -- Periscope saves the interrupt vectors it uses when activated, then restores them when done
- You can safely interrupt the system any time with the break-out switch, then continue running as if Periscope had never been used
- It doesn't use DOS except for file access, so you can debug your program even when DOS isn't working
- It protects itself; runaway programs can't touch the code residing in the Submarine board's memory, and Periscope I keeps tabs on its code in normal RAM to make sure it doesn't get corrupted

➧ Periscope II's code resides in normal RAM and is not protected nor recovered if corrupted

Periscope is COMPREHENSIVE because it gives you:
- A symbolic in-line assembler
- Over 75 breakpoint options, including sticky and temporary code breakpoints; breakpoints on register values, byte and word values, and reads and writes to ranges of memory and I/O ports using various tests; the ability to write your own breakpoint tests; and more
- Trace and traceback capability
- The ability to disassemble any location in memory, displaying the symbols, line numbers, and code from your source program
- The ability to search memory for string/byte patterns, procedure and address references, and

instructions
- The ability to view text files
- The ability to read and write disk files and absolute disk sectors
- The ability to read and write I/O ports
- The ability to compare two locations in memory, to move a block of memory from one location to another, and to make changes to memory
- The ability to jump over calls and interrupts
- The ability to display, change, save, and restore the registers and flags
- The ability to translate from hex to decimal and vice-versa, and to perform hex arithmetic
- User exits, so you can execute your own code from within Periscope

Periscope is FAST because:
- It's written entirely in assembler.

Symbols -- Your Road Map

Periscope is symbolic, meaning it allows you to use data and procedure names from your source program when you're debugging. This symbolic capability speeds 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 remember the location of code or data -- you can access it by name! You can use source line numbers and even actual source code if your compiler provides the information Periscope needs!

For example, assume that a program you're debugging calls a subroutine named PRINT_LINE, and you want to go to the first call of this subroutine in your program. You would enter 'G @PRINT_LINE'. ('G' is the Periscope Go command; '@' or '.' precedes any symbolic reference; see Chapter VI.) If symbols were not available, you'd have to know the address of the subroutine in order to get to it. If you were to disassemble this same program, the disassembly would display 'PRINT_LINE' wherever it is referenced in the program.

Symbols are read from the MAP file generated by your linker. Periscope supports the IBM, Microsoft, Phoenix, and DRI linkers. If your compiler generates line number references in the object files it produces, Periscope will give you line numbers and source code, as well. For more specific information on symbols, see the sections in Chapter VIII on the TS.COM and PUBLIC.COM utility programs and the description of RUN.COM in Chapter VII.

System Requirements

The system requirements for Periscope are:

- IBM Personal Computer, XT, AT, Compaq, or other close compatible such as Zenith Z-150, Columbia, Leading Edge, Sperry and many others.
- PC-DOS/MS-DOS 2.00 or later
- 128K RAM
- 1 disk drive
- 80-column monitor

The Submarine memory board may be installed in an expansion chassis if desired.

Periscope supports IBM's Enhanced Graphics Adapter. (See Chapter IX.)

II -- Backing Up The Disk

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 non-essential code and data, including the screen buffers, symbol table, and help file.

➧ For Periscope II, all code and data areas are placed in normal RAM.

PS.DEF -- This ASCII text file contains some sample record definitions that are read when RUN.COM is used to load a program.

PSDEMO.COM -- This program is a billboard-style demonstration of Periscope. It is not copyrighted and may be freely distributed.

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

PSHELP2.TXT -- This is the short form of the help file. It gives the syntax and a short description for each command. To use this file instead of the standard help file, rename it to PSHELP.TXT.

PSPATCH.COM -- This program is used to patch PS.COM to work on some of the less compatible computer systems. It is not needed if you use an IBM, Compaq, or other 99.44% compatible.

PUBLIC.COM -- This program is used to generate

public statements for most data items and procedures in an assembly program -- giving you the best possible symbol support for debugging.

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

RS.COM -- This program is used to verify a record definition (DEF) file and determine the record table size required by the 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 program or data file into memory, read the program's symbol table and record definition table if available and pass control to the resident debugger. This program will not work unless the Periscope software has been installed.

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

SYMLOAD.COM -- This program lets your program change Periscope's symbol table while your program is running. It is useful for programs that manage their own overlays or are running under another environment, such as TopView.

TS.COM -- This program is used to verify a MAP file and determine the symbol table size required by the MAP file. It enables you to efficiently allocate space for the symbol table and to generate a Periscope symbol file (PSS). This symbol file is optional for the standard linker (IBM/Microsoft), but is required for the Phoenix and DRI linkers.

USEREXIT.ASM -- This sample program illustrates user breakpoints and user exits from Periscope, including a DOS availability test and a display of the 8087/80287 status.

USEREXIT.COM -- This is the executable equivalent of USEREXIT.ASM.

III -- Equipment Installation

This chapter describes the installation of the Periscope I protected memory board and the installation of the Periscope II break-out switch.

The following topics are covered in this chapter.

- Checking for conflicts with memory and port use
- Setting the DIP switches
- Installing the Submarine board
- Installing the Periscope II break-out switch

➧ If you're using Periscope II, skip to the last section in this chapter.

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 device. 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 conflicts with IBM's Enhanced Graphics Adapter (EGA). If you have an EGA, try using memory at segment C400H. Submarine's default setting may also conflict with other 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. The use of these two ports conflicts with the 3Com Ethernet card, which uses ports 300H through 30FH. If you have this card, try using port 310H. Other expansion options may also use Submarine's default 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 conflicts with the memory or I/O ports used by the board, skip the next section on setting the DIP switches and proceed to the section titled Installing the Submarine Board.

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 four-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 100H and others used by expansion cards in your system. If port 300H is not available, try 310H. Consult the IBM Technical Reference Manual and the documentation for your non-IBM expansion cards to avoid conflicts.

The switches can be read by laying the board on a table, component (chip) 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 following table 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)

	'Hundreds'		'Tens'				'Units'	
Starting Port	S8	S7	S6	S5	S4	S3	S2	S1
00Ø	ON	ON	ON	ON	ON	ON	ON	ON
10Ø	ON	OFF	ON	ON	ON	ON	ON	ON
20Ø	OFF	ON	ON	ON	ON	ON	ON	ON
30Ø	OFF	OFF	ON	ON	ON	ON	ON	ON
30Ø	OFF	OFF	ON	ON	ON	ON	ON	ON
31Ø	OFF	OFF	ON	ON	ON	OFF	ON	ON
32Ø	OFF	OFF	ON	ON	OFF	ON	ON	ON
33Ø	OFF	OFF	ON	ON	OFF	OFF	ON	ON
34Ø	OFF	OFF	ON	OFF	ON	ON	ON	ON
35Ø	OFF	OFF	ON	OFF	ON	OFF	ON	ON
36Ø	OFF	OFF	ON	OFF	OFF	ON	ON	ON
37Ø	OFF	OFF	ON	OFF	OFF	OFF	ON	ON
38Ø	OFF	OFF	OFF	ON	ON	ON	ON	ON
39Ø	OFF	OFF	OFF	ON	ON	OFF	ON	ON
3AØ	OFF	OFF	OFF	ON	OFF	ON	ON	ON
3BØ	OFF	OFF	OFF	ON	OFF	OFF	ON	ON
3CØ	OFF	OFF	OFF	OFF	ON	ON	ON	ON
3DØ	OFF	OFF	OFF	OFF	ON	OFF	ON	ON
3EØ	OFF	OFF	OFF	OFF	OFF	ON	ON	ON
3FØ	OFF	OFF	OFF	OFF	OFF	OFF	ON	ON
30Ø	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, but not including x000:0000, where x is the number of 64K banks of RAM installed; B000:0000 to B000:0FFF if a monochrome adapter is installed; B800:0000 to B800:3FFF if a color/graphics adapter is installed; A000:0000 to C000:3FFF if an EGA 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 (chip) 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.

If memory starting at segment C000H is already in use, try using C400H. 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 following table 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	64 K Boundary S6	S5	S4	S3	16 K Boundary S2	S1
0000:0000 (0 K)	ON	ON	ON	ON	ON	ON
1000:0000 (64 K)	ON	ON	ON	OFF	ON	ON
2000:0000 (128 K)	ON	ON	OFF	ON	ON	ON
3000:0000 (192 K)	ON	ON	OFF	OFF	ON	ON
4000:0000 (256 K)	ON	OFF	ON	ON	ON	ON
5000:0000 (320 K)	ON	OFF	ON	OFF	ON	ON
6000:0000 (384 K)	ON	OFF	OFF	ON	ON	ON
7000:0000 (448 K)	ON	OFF	OFF	OFF	ON	ON
8000:0000 (512 K)	OFF	ON	ON	ON	ON	ON
9000:0000 (576 K)	OFF	ON	ON	OFF	ON	ON
A000:0000 (640 K)	OFF	ON	OFF	ON	ON	ON
B000:0000 (704 K)	OFF	ON	OFF	OFF	ON	ON
C000:0000 (768 K)	OFF	OFF	ON	ON	ON	ON
D000:0000 (832 K)	OFF	OFF	ON	OFF	ON	ON
E000:0000 (896 K)	OFF	OFF	OFF	ON	ON	ON
F000:0000 (960 K)	OFF	OFF	OFF	OFF	ON	ON
C000:0000 (768 K)	OFF	OFF	ON	ON	ON	ON
C400:0000 (784 K)	OFF	OFF	ON	ON	ON	OFF
C800:0000 (800 K)	OFF	OFF	ON	ON	OFF	ON
CC00:0000 (816 K)	OFF	OFF	ON	ON	OFF	OFF

Installing The Submarine Board

Before installing the board, be sure that the power is off and that the power cord is removed from the PC! To complete the installation, you'll need a small screwdriver.

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 break-out 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 break-out 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 break-out 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. If your system is other than an IBM, Compaq, or 99.44% compatible, run the program PSPATCH.COM. This program patches PS.COM to use addresses for interrupt vectors as found on your system at boot time. PSPATCH should be run without

any memory resident routines (including device drivers) loaded. If you have any problems, see the description of PSPATCH in Chapter VIII.

Step 8 -- Install Periscope by executing PS.COM. If you changed the DIP switches, you'll need to specify the memory and/or port settings as installation options. For example, if the memory was changed to C400:0000 and the port was changed to 304H, enter 'PS /M:C400 /P:304'. See Chapter V for more information on the installation options. If an error occurs, see Appendix A for an explanation of the error.

Step 9 -- After installing Periscope, press the remote break-out switch or the switch located on the board's mounting bracket. The debugger screen should be displayed, showing the current values of the registers and a disassembly of the current instruction. To continue, enter 'G' and press return.

Do not press the break-out switch before PS.COM is installed -- you'll get a parity error and have to turn the power off and back on.

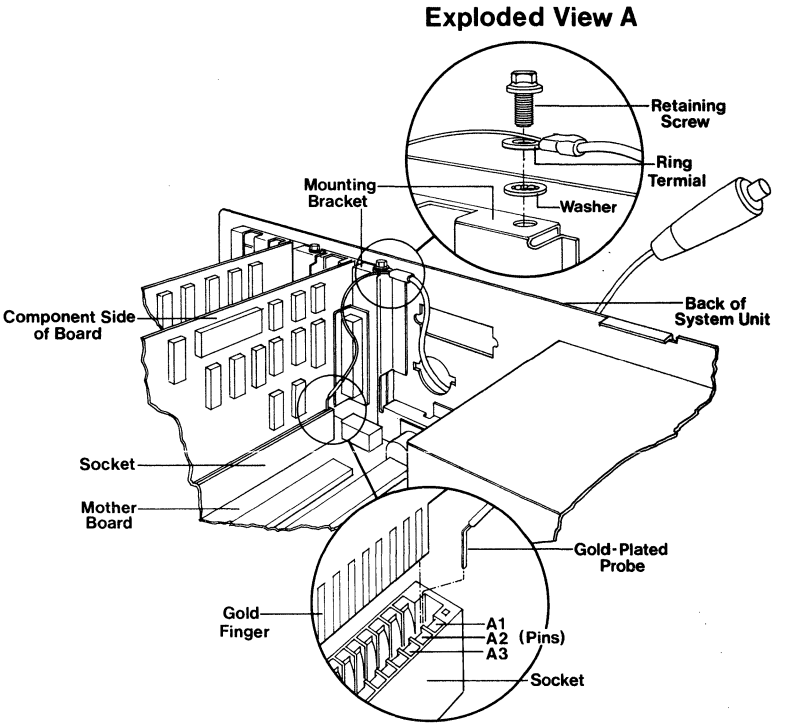
If the break-out switch doesn't work when Periscope is first installed, check the DIP switch setting for the 8087. If you have an 8087 installed, the switch should be OFF. If you don't have an 8087, the switch should be ON.

Installing The Periscope II Break-Out Switch

Before installing the break-out switch, be sure that the power is off and that the power cord is removed from the PC! To complete the installation, you'll need a small screwdriver and a bright light source, such as a flashlight. Please refer to the illustration below. It shows the switch installed in an IBM PC. Keep in mind that there may be slight physical differences if you're using another machine.

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 cable assembly has a push-button switch, a five foot length of cable and two connectors. One of the connectors is a ring terminal (see Exploded View A) and the other is a gold-plated probe (see Exploded View B).



Route the connector end of the cable assembly into the PC from the back of the system unit. There are several possible ways of routing the cable, either through a knock-out panel or in the space between the keyboard connector and the expansion slots. Do NOT install the cable so that it is lying on top of the back panel -- when the cover is installed, the cable may be crimped and possibly damaged.

Step 3 -- Remove the retaining screw on the expansion slot mounting bracket nearest to the power supply. This slot is usually, but not always, occupied by a disk controller card. Note -- If this slot is not in use, use the in-use slot that gives you the best accessibility to the component (chip) side of the board.

Insert the retaining screw through the ring terminal and then place the washer on the retaining screw. Re-install the retaining screw as shown in the illustration. Align the cable so that it is parallel with the back panel of the system unit. Be sure it has a minimum of twists and turns between the ring terminal and the point where the cable comes into the system unit.

The ring terminal provides both an electrical ground and strain relief. Be sure that it is securely installed!

Step 4 -- Using the flashlight or other bright light source, install the gold-plated probe into pin A1 of the expansion slot used in Step 3. The slot must be in use for the probe to be attached securely.

Pin A1 is the pin on the component (chip) side of the expansion board that is closest to the board's mounting bracket. This pin is used to generate a Non-Maskable Interrupt (NMI).

To install the probe, hold it so that the probe is pointing downward and the cable is angled away from the board. Push the probe down firmly into pin A1 between the gold finger on the board and the connector in the socket as shown in the illustration. Note -- not all boards have a gold finger at pin A1 -- look for the first socket connector to positively identify the pin. Push the probe in as far as possible to ensure a good connection and to keep the uninsulated part of the probe from contacting anything other than the desired pin.

Step 5 -- Double check the placement of the probe. It should be in the pin on the component (chip) side

of the expansion board nearest the board's mounting bracket. The probe must be between the board and the connector pin in the socket -- it must not be between the connector pin and the outer edge of the socket!

Some sockets have an extra dummy hole at the end of the socket. Be sure not to insert the probe in this hole!

Step 6 -- Double check the placement of the ring terminal. It should be firmly held by the retaining screw that holds the expansion board in place. For the best electrical contact, be sure that the supplied washer has been installed between the ring terminal and the board's mounting bracket.

Step 7 -- Slide the cover of the system unit back over the machine and install the cover mounting screws.

Step 8 -- Re-connect all peripherals and replace the power cord.

Step 9 -- Boot the system. If your system is other than an IBM, Compaq, or 99.44% compatible, run the program PSPATCH.COM. This program patches PS.COM to use addresses for interrupt vectors as found on your system at boot time. PSPATCH should be run without any memory resident routines (including device drivers) loaded. If you have any problems, see the description of PSPATCH in Chapter VIII.

Step 10 -- Install Periscope by executing PS.COM. If an error occurs, see Appendix A for an explanation of the error.

Step 11 -- After installing Periscope, press the break-out switch. The debugger screen should be displayed, showing the current values of the registers and a disassembly of the current instruction. To continue, enter 'G' and press return.

Do not press the break-out switch before PS.COM is installed -- you'll get a parity error and have to turn the power off and back on.

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.

IV -- Tutorial: Up Periscope

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

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 Periscope into 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.

➧ For Periscope II, you don't have to install the break-out switch before taking the tutorial.

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:

- PS.DEF -- Contains record definitions for the PSP (Program Segment Prefix) and the FCB (File Control Block) and other records

- SAMPLE.ASM -- The source code for SAMPLE.COM

- SAMPLE.COM -- The executable program

- SAMPLE.MAP -- The MAP file produced by the linker when the /M option is used

Periscope uses the DEF file to read keyboard definitions and record definitions into memory. This file is created with an editor as a standard ASCII text file. In the file PS.DEF, there are two sample keyboard definitions and four record definitions. These record definitions can be used to display any area of memory in an easy-to-read format. There's a utility program to verify and determine the amount of memory required by a DEF file. See the description of RS.COM in Chapter VIII for more information.

Periscope uses the MAP file in order to replace the RAM addresses normally supplied while debugging with the more meaningful labels used in the program being debugged. The MAP file can be created when you link your program by specifying a MAP file and the '/M' and '/LI' options. This file contains the public

code and data addresses and their labels. Periscope then uses these symbols to display the labels rather than the numeric addresses. The more symbols you have, the easier it is to debug your program -- so we've provided a program to generate as many PUBLIC statements as possible. Also, there's a program to verify and size MAP files and convert non-standard MAP files to Periscope's format. See the descriptions of PUBLIC.COM and TS.COM in Chapter VIII for more information.

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 START and ends at DOSRET. The mainline calls three procedures. The first procedure, GETMEM, is called once to retrieve the total memory and the available memory. CONVERT is called twice to convert the memory size from hex to ASCII. Finally, DISPLAY 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, record definitions, and key definitions were loaded into the symbol tables, record tables, and key definition 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     START
```

Registers BX and CX show the size of the program. Registers DS, ES, SS, and CS all show the PSP segment 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 START, not an offset. If you need to know the offset, enter 'U @START' to disassemble memory starting at the symbol START.

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 'DB CS:0'. This shows the first 128 bytes of the PSP in Byte format. For a more useful display, enter 'DR CS:0 @PSP'. The record definition makes it much easier to see what's what in the PSP. You can add record definitions as you need them by editing the DEF file (see the description of RS.COM in Chapter VIII). 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 START. Enter 'U' to disassemble the next few lines --

```
                START:
0C73:0137  E81700       CALL    GETMEM
0C73:013A  A13301       MOV     AX,[TOTMEM]
0C73:013D  BF1001       MOV     DI,0110                 ; TMEMORY
0C73:0140  E82400       CALL    CONVERT
0C73:0143  A13501       MOV     AX,[FREMEM]
0C73:0146  BF2C01       MOV     DI,012C                 ; AMEMORY
0C73:0149  E81B00       CALL    CONVERT
0C73:014C  E83400       CALL    DISPLAY
                DOSRET:
0C73:014F  CD20         INT     20
                GETMEM:
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 SA (Search Address) command is used to search for address references. Enter 'SA @START @DISPLAY @CONVERT' to search from START to DISPLAY for references to the procedure CONVERT. Also, try 'SA @START @DISPLAY @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, DOSRET. To do this enter 'BC @DOSRET' and press return.

To see the symbols available, press F8. Now, 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  DOSRET
BW  TOTMEM NE ØØØØ
```

Now enter 'GT' 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=Ø1ØØ  BX=ØØØØ  CX=ØØØ6  DX=ØØØØ  SP=FFFC  BP=ØØØØ  SI=ØØØ2  DI=ØØØØ
DS=ØC73  ES=ØC73  SS=ØC73  CS=ØC73  IP=Ø15D   NV UP EI PL NZ NA PE NC
ØC73:Ø15D  8CCB          MOV     BX,CS
```

To see the instruction that caused the breakpoint, enter 'TB'. Periscope's traceback command shows previously executed instructions. 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 'DI @TOTMEM L2'. This is the total memory in K -- note that the result is in decimal, not hex. To show the value in hex, use 'DW @TOTMEM L2'. To convert back to decimal, use the translate command. Enter 'X nnnn', where nnnn is the hex 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 'RET' instruction, use the 'J' command one more time to get back to the mainline code.

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

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

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

To view the source file for this program, enter 'V SAMPLE.ASM' and press return. Use the PgUp, PgDn, Home, End, Up, and Down arrow keys to move through the file. When done, press the Esc key to return to the debugger prompt.

To set Periscope's windows, enter '/W DRSU'. This sets up four windows showing data, register, stack, and disassembly information. The presence, order, and length of these windows can be changed on the fly. Try '/W D:8 R U:8'. Now turn the windows off again by using '/W'.

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

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 entering the number after the register -- try entering 'R CX 10'. Afterwards, check the result -- enter 'R' and press return to display the registers.

Display memory at offset 200H by entering 'DB 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. The result might be:

```
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 quit options are then displayed. Enter 'R' to return to DOS or 'C' to continue and then press return. Either response ends the debugging session.

V -- Installing The Periscope Software

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

- Installation options and defaults
- Alternate start-up methods

Installation Options

To load Periscope, enter 'PS' from the DOS prompt, followed by the desired installation options. (See the next section in this chapter for alternate start-up methods). Periscope has the following default values (in alphabetical order by installation option):

A -- One monitor is assumed.
B -- The traceback buffer is presumed to be 1K (enough for 32 entries).
C -- The screen color is low-intensity white on black.
D -- The original INT 13H (disk) vector is not restored before a short boot.
E -- The source file buffer is presumed to be 1K.
F -- Slow color output is presumed.
H -- On-line help is not available.
I -- No user interrupt exit is available.
J -- Sys Req does not activate Periscope.
K -- Shift-PrtSc does not activate Periscope.
L -- Periscope's external tables are loaded as low in memory as possible.
M -- The write-protected memory begins at C000:0000 and ends at C000:3FFF.
P -- The write-protect ports are 300H and 301H.
R -- The record table size is presumed to be 1K.
S -- The original screen size is presumed to be 4K, which is sufficient for text mode only.
T -- The symbol table size is presumed to be 1K.
V -- The BIOS interrupt vectors used by Periscope (8H, 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.
W -- Windows are turned off.
Z -- External tables are not suppressed.

➧ For Periscope II, the M, P, and Z installation options are not available.

Use the installation options described below to

change the above defaults. All options contain a slash (/) and a single letter mnemonic. Some of the options include a number. If a number is used, it is always preceded by a colon (:) and is always in hexadecimal notation.

The installation options are:

? -- Display help information about Periscope's installation options.

/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 the break-out switch is pressed or when a program is loaded with RUN.COM. If this option is used, the /S option is ignored and no memory is reserved for the original or debug screen buffer.

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

/B:nn -- Set the size of the traceback buffer to something other than 1K. This option is used when you want more than the 32 traceback entries available from the default buffer size. The one or two-digit hexadecimal number nn is the number of K desired. The number may be from zero to 10H K. Remember that the input is in hex!

/C:nn -- This option sets the color attribute for Periscope's display. The two digit number is 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 table below.

The layout of the color attribute bits is:

Bit number	7	6	5	4	3	2	1	Ø
Use	X	R	G	B	H	R	G	B
		background color				foreground 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

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 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.

/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.

/E:nn -- Set the size of the source file buffer to something other than 1K. This option is used to improve performance when the Unassemble Source or View file commands are used. The one or two-digit hexadecimal number nn is the number of K desired. The number may be from zero to 10H K. Remember that the input is in hex!

/F -- Set fast output for a color monitor. This option should be used if you have a 'snow-free' color card. The original IBM color card suffers from 'snow' when you attempt to write directly to the screen buffer. A snow-free card allows you to write directly to the screen without any interference. If you have one of these snow-free cards, use this option to speed up the screen display.

/H -- Install the on-line help. 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 current directory.

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. Two versions

of Periscope's help file are provided. PSHELP.TXT is the large version, containing syntax requirements and descriptions for each of Periscope's commands. PSHELP2.TXT is a much smaller version, containing syntax requirements and a short description for each command. To use the smaller help file, rename it to PSHELP.TXT and use the /H installation option.

/I:nn -- This option is used to allow access to user-written code from Periscope. The program must be a memory-resident routine that is already installed using an interrupt from 60H to FFH. It must meet the specifications as defined for the program USEREXIT.ASM in Chapter VIII. The two-digit hexadecimal number nn must be from 60H to FFH.

/J -- This option enables the Sys Req key on the AT as an entry to Periscope. Sys Req will not work when interrupts are disabled -- try using the break-out switch in this case.

Note that the CS:IP shown after using Sys Req to enter Periscope is the instruction after INT 15H in the keyboard handler (INT 9). To make PS.COM reloadable, this option does not pass interrupts on through to a previously installed interrupt handler. If you're using INT 15H for something else, do not use this option. Instead, modify your INT 15H handler to generate an INT 2 when Sys Req is pressed. Once this option is used, the use of Sys Req will activate Periscope, until the system is re-booted using a normal boot, since DOS does not modify INT 15H.

/K -- This option enables the Shift-PrtSc keys as an entry to Periscope. 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 the instruction after INT 5 in the keyboard handler (INT 9). Once this 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 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!

/L:nnnn -- Load Periscope's tables starting at the specified segment. Normally, Periscope's external tables and code are loaded as low as possible (just

after DOS). If you're debugging device drivers or other non-DOS programs, this option can be used to place Periscope's tables in an area of memory that is not corrupted by a short boot. The four-digit hexadecimal number is the segment where the tables should start. It must be greater than the current PSP plus 10H paragraphs and less than the top of memory minus 1000H paragraphs. For example, if the PSP is C00H and the top of memory is 5000H, the limits for this option would be C10H through 4000H.

/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.

➧ The '/M' installation option is not available for Periscope II.

/P:nnn -- Set the protected memory ports to something other than 300H and 301H. The three-digit hexadecimal number nnn 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.

The architecture of the 8086 family supports 64K I/O ports (0 through FFFFH), but the IBM PC and compatibles support only the first 1024 (3FFH) of these ports, many of which are reserved. The high 6 bits are ignored, with the result that port 1200H is really 200H, etc.

➧ The '/P' installation option is not available for Periscope II.

/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 zero to 20H K. Remember that the input is in hex! See the description in Chapter VIII of RS.COM, which 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 ignored if the /A option is used. It is used when debugging programs that use the color-graphics adapter. It is only required for single-monitor systems where a 4K screen buffer is too small. The one or two-digit hexadecimal number nn is the number of K desired, from zero to 20H K. 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 the number as small as possible, since each Trace or Go command has to copy this buffer twice.

/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 zero to 3FH (63) K. Remember that the input is in hex! See Chapter VIII for information on TS.COM, which determines the symbol table size required for a MAP file and generates compact memory-image symbol files.

/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 their power-on default values. This is done to make Periscope as dependable as possible. When Periscope is exited, the interrupt vectors are restored to their values on entry. 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 8 (timer), 9 (keyboard), 10H (video), 16H (keyboard I/O), 17H (printer), and 1CH (timer control). Note that each vector must be entered separately. For example, to leave vectors 10H and 17H alone, use '/V:10 /V:17'. Periscope temporarily changes the Ctrl-Break vector (1BH), the DOS Ctrl-Break exit address (23H), and the DOS Fatal error vector (24H), but these changes cannot be overridden.

/W -- This option is used to set Periscope's windows. Periscope can window data, stack, register, and/or disassembly information. The window specification can be entered when PS.COM is run or from within Periscope. 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 window specification 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 in the input line.

The 'D' window shows data in any of the display formats. The 'R' window shows register and flag information. The 'S' window shows the current stack. The 'U' window shows disassembled instructions.

':nn' defines the length of the window (in hex as with all other installation options). 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 (16)
Register	2	2	2
Stack	2	1	10H (16)
Unasm	4	4	10H (16)

/Z -- This option is used to keep Periscope from using any memory external to the Submarine board. It overrides other options that allocate memory outside of the 16K on the board. This option should be used only when necessary, since it greatly restricts Periscope's capabilities.

➧ The '/Z' installation option is not available for Periscope II.

The installation options can be entered in any combination of upper and lower case. No spaces are required between entries, except after numbers in the window specification. Some examples follow:

PS /A /W D:8 R U:8 -- Use two monitors and establish windows showing eight lines of data, two lines

(default) of register information, and eight lines of disassembly.

PS /M:A000 /P:31C /S:10 -- 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. Reserve 16K to save the program's screen on a single-monitor 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 external data areas can be from zero K (if the /Z option is used) to over 128K. These buffers are located in normal RAM using the terminate and stay resident function of DOS. The protected memory on the Submarine board is used for the critical code and 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 an explanation 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 if you want to ensure its presence each time the system is booted. If you are not sure whether Periscope is installed, do not press the break-out switch -- try running RUN.COM instead. If Periscope is not installed, RUN will display an error message.

Alternate Start-up Methods

Periscope can be installed via a full-screen display that lets you choose among all possible options. To use this method, enter 'PS *' when the DOS prompt is displayed. This full-screen method is self-explanatory. All 'information' messages are suppressed when this screen is used -- only error messages are displayed. When function key F9 is used, this full-screen display generates a response file named PS that can later be used to start Periscope with 'PS @PS'.

Normally, we recommend that you invoke PS.COM from an AUTOEXEC.BAT file. If you sometimes need different Periscope options for debugging different types of programs, the response file is for you. This file is an ASCII text file that contains any

PS.COM installation options. For example, if you create a file named C:STD that contains '/b:4 /v:10 /a', you can enter 'PS @C:STD' to load Periscope using the options found in the file named C:STD. You can use 'PS /C:17 @C:STD' to set the color attribute and then retrieve the options from the file named C:STD. Any options entered after a response file name are ignored. For example, 'PS @C:STD /C:17' would not set the color attribute. The file name used for a response file may be any legal file name.

VI -- Using Periscope

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

- The quit options (Boot, Continue, Debug, Return to DOS, Short boot)
- Keyboard usage
- Debugger parameters
- The debug commands

The Quit Options

When you use 'Q' to quit the debugger, Periscope's quit options are displayed. The prompt is:

(XX) Normal boot (B), Continue (C), Debug (D), Return to DOS (R), or Short boot (S)? _

The first value in parentheses indicates the method that was used to activate Periscope. The possible values are:

BR -- A monitor breakpoint was taken.
GO -- A code breakpoint was taken.
P1 -- Parity error 1 (motherboard) occurred.
P2 -- Parity error 2 (expansion memory) occurred.
SW -- The break-out switch was pressed. Note: this also appears for Shift-PrtSc, Sys Req, and 80286 exception interrupts 6 and DH.
TR -- An instruction was traced.

➧ When the Periscope II break-out switch is used, 'P2' is shown instead of 'SW'.

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 a critical area 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. This method does not set any breakpoints.

The debug (D) option is used to return to the

debugger.

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 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 64K are overwritten by the boot record and DOS. This can cause problems for some memory-resident programs, such as a low-memory RAM disk.

Keyboard Usage

Since Periscope uses DOS functions only for the Load, Name, View, Unassemble Source, and Write commands, 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 Periscope.

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

F1 or Right Arrow -- copies one character from the previous command line into the current command line (same as DOS).

F3 -- copies the remainder of the previous command line into the current command line. This key copies up to, but does not include 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 -- cancels the current command line (same as DOS).

Backspace or Left Arrow -- deletes the current keystroke and backs up one character

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). Any control codes or special characters other than carriage return and line feed are suppressed. Only the non-windowed area of the screen is printed.

Ctrl-S -- suspends output until another key is pressed. Use this key combination to keep information from scrolling off the screen too quickly.

F6 -- pauses when the screen is full. To change the pause state, 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.

F7 -- displays the current record definitions as read from a DEF file when RUN is started. If the cursor is at the beginning of a command line, all record definitions are displayed. You can display record definitions that start with a character sequence by entering the desired characters and then pressing F7. For example, to display all record definitions starting with 'PS', enter 'PS' at the start of a command line and press F7. Be sure not to enter any spaces before or after the search name.

F8 -- displays the address and name of the symbol table entries as read from a MAP or PSS file when RUN is started. If the cursor is at the beginning of a command line, all symbols are displayed. You can display symbols that start with a character sequence by entering the desired characters and then pressing F8. For example, to display all symbols starting with the letter 'A', enter 'A' at the start of a command line and press F8. Be sure not to enter any spaces before or after the search name.

F9 -- same as Ctrl-S.

F10 -- switches from the debugger screen to the program's screen if only one monitor is being used.

If two monitors are used and the /A option was used to install PS.COM, this key has no effect. To return to the debugger screen from the program's screen, press any key.

Semi-colon -- This character is used as a pseudo carriage-return. Use it to enter multiple commands on one line. For example, if you're tracing through a program that requires repetitive Go and Fill commands, you could enter 'G @NEWPAGE;F @PAGENO L2 0' to go to the line labeled NEWPAGE and fill memory starting at PAGENO. 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), except if the /K option was used when PS.COM was installed. Be careful if control codes have been displayed on the screen with the Display or Xlate commands -- use Ctrl-PrtSc to avoid output of control codes to the printer.

Alt-F1 through Alt-F10 -- saves the current command line of up to 64 characters. Enter a command and press Alt and a function key to save the command for later recall. To get a carriage return after the saved command, enter a semi-colon as the last character of the command before saving it. Key assignments may be read from the DEF file -- see the description of RS.COM in Chapter VIII.

Ctrl-F1 through Ctrl-F10 -- recalls the command line saved by Alt-Fn. The recall function can be used anywhere within a command. To easily remember the key usage, think of Alt-Fn as 'A'ssign and Ctrl-Fn as 'C'all.

Debugger Parameters

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

Each command requires at least a single-character mnemonic. All but a few commands require additional input.

The various parameters used by Periscope are defined below, in alphabetical order. Brackets ([ ]) in the command syntax are used to indicate an optional entry. (Note that brackets actually entered in a

command line are used to indicate that the address is to be used as a near pointer.) An ellipsis (...) is used to indicate a repetitive entry.

$ -- The dollar sign or 'here' indicator can be used with the Display commands to replace the display address and more easily display some types of data. It assumes a value equal to one more than the last byte previously displayed. For example, if you want to page through memory displaying 200H bytes at a time, you can use 'D $ L200' rather than having to specify an address each time. Similarly, the DR command can be used to display repeating record definitions. For example, 'DR $ .RECORD' can be used to display a repeating fixed-length record.

<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 or an 8-bit register.

<decimal number> -- A decimal number from 0 to 65535. 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 (=Ø)
Overflow	OV	NV
Direction	DN (STD)	UP (CLD)
Interrupt	EI (STI)	DI (CLI)
Sign	NG (negative)	PL (positive)
Zero	ZR (zero)	NZ (non-zero)
Auxiliary carry	AC	NA
Parity	PE (even)	PO (odd)
Carry	CY (STC)	NC (CLC)

<function> -- The debugger command, such as D

(Display memory), or U (Unassemble).

<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). For example "03 'COMMAND COM' 12 34" is a list composed of a byte, a string, and two trailing bytes.

<name> -- A file name, including drive, path, and extension as needed.

<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 if they both reference the same segment and if the offset of the second symbol is greater than or equal to the offset of the first symbol.

<register> -- A machine register. The 16-bit registers are AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, and IP. The 8-bit registers are AH, AL, BH, BL, CH, CL, DH, and DL.

<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 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 quote to delimit the string, or enter two quotes where the single embedded quote is desired.

<sub-function> -- The mnemonic used with most 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 the current 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 PSS or MAP file when the corresponding program is loaded by RUN. On entry, a symbol is always preceded by '@' or a period. 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).

[ ] -- Brackets around an address are used to indicate that the offset is to be used as a near pointer to another offset within the specified segment. The trailing bracket is optional. For example, if the word at CS:250H contains 1234H, 'U [CS:250]' disassembles memory starting at CS:1234.

{ } -- Braces around an address are used to indicate that the segment and offset are to be used as a far pointer to another segment and offset pair. The trailing brace is optional. For example, to disassemble INT 10H, enter 'U {0:40}'. This command uses the offset at 0:40H and the segment at 0:42H, which is interrupt vector 10H.

The Debugger Commands

The debugger commands are described on the following pages.

The code and data areas needed to execute essential Periscope commands are stored in the protected memory. These commands are always available and show a type of Internal in the following descriptions. The code needed to execute non-critical commands and the error messages are stored in normal RAM (unless the /Z option is used when Periscope is installed). These commands show a type of External in the following descriptions. Before executing an external command, Periscope performs a checksum to verify that the memory has not been modified. If the memory has been corrupted, Error 26 is displayed. To reload a corrupted external code area, execute RUN.COM.

➧ For Periscope II, all of Periscope's code and data areas are in normal RAM, so all commands are considered External. No checksum or reload facilities are available for Periscope II, so it is possible for its code to be corrupted by a runaway program.

Command: Help

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

Type: External

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. If external functions are not available, Error 26 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: Assemble to memory

Syntax: A [<address>]

Type: External

Description: This command assembles instructions to memory.

To use the in-line assembler, enter 'A [<address>]' when Periscope's prompt is displayed and press return. The assemble address is then displayed. If no address is specified, CS:IP is used. Enter the instructions to be assembled and press return. To terminate the assembly, press return when the cursor is at the beginning of a new line.

The assembler supports all of the 8086, 8088, 80186, and real-mode 80286 opcodes. The protected-mode opcodes of the 80286 are not supported. If a prefix instruction such as a segment override or a repeat prefix is used, it must be on a separate line preceding the instruction it affects. Various forms of the opcodes are supported, including synonyms such as JE and JZ, etc. There are two special cases -- string primitives such as MOVS must explicitly reference a byte or word (MOVSB or MOVSW) and a far return must be entered as RETF.

Jump or call instructions generate the shortest form of call for the address specified. When referencing memory, be sure to use brackets around the address field to differentiate it from a direct reference.

When using symbols, the symbol name must be preceded by '@' or a period. If you are referencing the contents of a symbol, be sure to put the symbol name in brackets -- e.g., MOV AX,[@PAGENO]. To get the offset of a symbol into a register, do not use the brackets -- e.g., MOV AX,@PAGENO. Symbols may also be used as arguments to JMPs and CALLs.

For instructions that require the phrase 'byte ptr' or 'word ptr' to specify the width of the operation, use 'b' or 'w', in upper or lower case, instead.

Example: To assemble an instruction at 1234:5678 to jump to the symbol @NEW_PAGE, enter 'A 1234:5678' and press return. Then enter 'JMP @NEW_PAGE' and press return. Press return again to exit the in-line assembler.

Command: Assemble then Unassemble

Syntax: AU [<address>]

Type: External

Description: This command is the same as the Assemble command described previously, except that it disassembles an instruction immediately after assembling it.

This immediate feedback was originally used to debug the in-line assembler. We left it in Periscope since users reacted positively to it.

Example: To assemble an instruction at CS:IP to move the value of the symbol .TOTMEM to register AX, enter 'AU' and press return. Then enter 'MOV AX,[.TOTMEM]' and press return. The instruction is disassembled and then the next prompt is displayed. Press return again to exit the in-line assembler.

Command: Display Breakpoints

Syntax: BA ?, BB ?, BC ?, BI ?, BL ?, BM ?, BP ?, BR ?, BU ?, or BW ?

Type: External

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

A leading minus sign indicates that the breakpoint is disabled. The sub-function indicates the breakpoint group to be displayed as follows:

A -- display All breakpoints
B -- display Byte breakpoints
C -- display Code breakpoints
I -- display the Interrupt breakpoint status
L -- display the source code Line breakpoint status
M -- display Memory breakpoints
P -- display Port breakpoints
R -- display Register breakpoints
U -- display User 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 *, BI *, BL *, BM *, BP *, BR *, BU *, or BW *

Type: External

Description: These commands clear the current breakpoints.

The monitor breakpoints and sticky code breakpoints are remembered until they are cleared. The sub-function indicates the breakpoint group to be cleared as follows:

A -- clear All breakpoints
B -- clear Byte breakpoints
C -- clear Code breakpoints
I -- clear the Interrupt breakpoint
L -- clear the source code Line breakpoint
M -- clear Memory breakpoints
P -- clear Port breakpoints
R -- clear Register breakpoints
U -- clear User 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: Enable Breakpoints

Syntax: BA +, BB +, BC +, BI +, BL +, BM +, BP +, BR +, BU +, or BW +

Type: External

Description: These commands enable the current breakpoints.

Enabled breakpoints are used when the G or GT command is entered. When enabled, the breakpoint display does not show a leading minus sign before the breakpoint. The sub-function indicates the breakpoint group to be enabled as follows:

A -- enable All breakpoints
B -- enable Byte breakpoints
C -- enable Code breakpoints
I -- enable the Interrupt breakpoint
L -- enable the source code Line breakpoint
M -- enable Memory breakpoints
P -- enable Port breakpoints
R -- enable Register breakpoints
U -- enable User breakpoints
W -- enable Word breakpoints

Examples:

Assume that a Byte and a Port breakpoint had been set and then disabled.

'BP +' enables the Port breakpoint(s).

'BB +' enables the Byte breakpoint(s).

'BA +' enables All breakpoints. Note that the Interrupt and Line breakpoints are enabled only if they have been previously turned on with 'BI +' or 'BL +' and then disabled.

Command: Disable Breakpoints

Syntax: BA -, BB -, BC -, BI -, BL -, BM -, BP -, BR -, BU -, or BW -

Type: External

Description: These commands disable the current breakpoints.

Disabled breakpoints are ignored when the G or GT commands are used. When RUN.COM is used, all breakpoints are disabled to prevent possible interference with the new program. Use the enable breakpoint command to enable the desired breakpoints.

When disabled, the breakpoint display shows a leading minus sign before the breakpoint. The sub-function indicates the breakpoint group to be disabled as follows:

A -- disable All breakpoints
B -- disable Byte breakpoints
C -- disable Code breakpoints
I -- disable the Interrupt breakpoint
L -- disable the source code Line breakpoint
M -- disable Memory breakpoints
P -- disable Port breakpoints
R -- disable Register breakpoints
U -- disable User breakpoints
W -- disable Word breakpoints

Examples:

Assume that a Byte and a Port breakpoint have been set.

'BP -' disables the Port breakpoint(s).

'BB -' disables the Byte breakpoint(s).

'BA -' disables All breakpoints. Note that the Interrupt and Line breakpoints are disabled only if they have been previously turned on.

Command: Breakpoint on Byte

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

Type: External

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, a breakpoint is taken. To trace execution with this breakpoint enabled, the 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, display, enable, and disable functions may also be present on the line. After being set, these breakpoints are remembered until they are cleared.

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> [...]

Type: External

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 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, display, enable, and disable 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 Interrupt

Syntax: BI +

Type: External

Description: This command is used to set a breakpoint when a software interrupt is executed.

This breakpoint is used to get to the next instruction that performs an interrupt. Use this breakpoint to watch the software interrupts performed by a program. After setting the Interrupt breakpoint, use GT to execute to the next interrupt.

Note that 'BI +' must be used to turn on Interrupt breakpoints for the first time -- 'BA +' (enable all breakpoints) will enable the Interrupt breakpoint only if it has been previously turned on and then disabled. After being set, this breakpoint is remembered until it is cleared.

Example: 'BI +' turns the Interrupt breakpoint on so that a subsequent GT command will stop when the next 'INT xx' instruction is reached.

Command: Breakpoint on Line

Syntax: BL +

Type: External

Description: This command is used to set a breakpoint when a source code line is executed.

This breakpoint is used to get to the next instruction that corresponds to a source line of a high-level language program. If your program is executing and you press the break-out switch, chances are very good that the program will be stopped in DOS, BIOS, or in a library routine. This breakpoint is a convenient method of getting back to the source program. It requires source line numbers to be in the symbol table -- symbols added with the ES command will not suffice. After setting the Line breakpoint, use GT to execute to the next source line.

Note that 'BL +' must be used to turn on Line breakpoints for the first time -- 'BA +' (enable all breakpoints) will enable the Line breakpoint only if it has been previously turned on and then disabled. After being set, this breakpoint is remembered until it is cleared.

Example: 'BL +' turns the Line breakpoint on so that a subsequent GT command will stop when the next instruction that corresponds to a source code line is reached.

Command: Breakpoint on Memory

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

Type: External

Description: This command is used to set a breakpoint when a range of memory will be executed, read and/or written.

The two addresses may have different segments, but the second address must not be lower 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. The 'C' breakpoint will occur only if CS:IP is in the specified range. The read or write breakpoints will occur only if a read or write starts in the specified range. If any of the tests pass, a breakpoint is taken. To trace execution with this breakpoint enabled, the GT command must be used. This breakpoint stops execution of a program on the instruction that will execute, read or write the specified range of memory. Multiple breakpoints may be set on a single input line. The breakpoint clear, display, enable, and disable functions may also be present on the line. After being set, these breakpoints are remembered until they are cleared.

Examples:

'BM @DATASTART @DATAEND W' sets a Memory breakpoint for memory from DATASTART thru DATAEND. Any instruction that writes to this range of memory causes 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 Port

Syntax: BP <port> <port> I and/or O [...]

Type: External

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

The second port must be greater than or equal to the first port. Up to eight breakpoints may be set at one time. A breakpoint will occur only if an IN or an OUT occurs to a port in the specified range. If any of the tests pass, a breakpoint is taken. To trace execution with this breakpoint enabled, the GT command must be used. This breakpoint stops execution of a program on the instruction that will read or write the specified range of ports. Multiple breakpoints may be set on a single input line. The breakpoint clear, display, enable, and disable functions may also be present on the line. After being set, these breakpoints are remembered until they are cleared.

Examples:

'BP 310 31F I' sets a Port breakpoint for ports from 310 to 31F. Any instruction that reads from this range of ports causes a breakpoint to be taken, before the instruction is executed.

'BP * 304 304 O ?' clears all Port breakpoints, sets a breakpoint to trap any writes to port 304, and displays the Port breakpoint.

Command: Breakpoint on Register

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

Type: External

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, a breakpoint is taken. To trace execution with this breakpoint enabled, the 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. Any of the 16-bit or 8-bit registers may be used. The breakpoint clear, display, enable, and disable functions may also be present on the line. After being set, these breakpoints are remembered until they are cleared.

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 User test

Syntax: BU <number> [...]

Type: External

Description: This command is used to enable a user-written breakpoint.

The User breakpoints permit breakpoint tests not provided by Periscope. The number may vary from 1 to 8, indicating one of eight possible User breakpoints. To use this breakpoint, a program similar to USEREXIT.ASM as described in Chapter VIII must be installed before PS.COM is run. Also, the /I installation option must be used when PS.COM is run. On return from the user routine, register AL is set to 1 if a breakpoint is to be taken. Any other value causes no breakpoint to be taken. Note that any other breakpoints currently set may cause a breakpoint to be taken.

The first User breakpoint in the sample program USEREXIT.ASM is used to set a breakpoint when DOS is available for file I/O. If you need to perform DOS functions after pressing the break-out switch, this User breakpoint will come in handy.

Multiple breakpoints may be set on a single input line. The breakpoint clear, display, enable, and disable functions may also be present on the line. After being set, these breakpoints are remembered until they are cleared.

Examples:

Assuming that a user-written interrupt handler has been installed using INT 60H and that PS.COM had the '/I:60' installation option, 'BU 1' enables User breakpoint number 1.

'BU 9' returns an error since the User breakpoint range is from one to eight.

Command: Breakpoint on Word

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

Type: External

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, a breakpoint is taken. To trace execution with this breakpoint enabled, the 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, display, enable, and disable functions may also be present on the line. After being set, these breakpoints are remembered until they are cleared.

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>

Type: Internal

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 current format

Syntax: D [<range>]

Type: Internal

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

When Periscope is installed, Display defaults to a Byte format. Subsequent Display commands use the most recent explicit format. See both the descriptions of the various display formats on the following pages as well as the information applicable to all display formats in the next paragraphs.

The syntax for all of the Display commands except DE and DR is very flexible. If you enter 'Dx', where x is the sub-function, memory is displayed starting where the last Display command left off. If you enter 'Dx <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 'Dx <number> <length>' the number is presumed to be an offset, and the segment is presumed to be DS.

When display information is not shown in a window and one or more lines in the middle of the display are found to be multiple 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.

Command: Display using ASCII format

Syntax: DA [<range>]

Type: Internal

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

Each line of the display shows the starting segment and offset and up to 64 bytes of ASCII characters. All characters are displayed as is, except for the control characters nul, backspace, tab, carriage return, and line feed. Nuls are converted to spaces and the other three control characters are converted to periods. A new line is started when a CR/LF is found. If a tab character is found, the output position is moved to the next tab stop.

For example, if you enter 'DA @TEXT' the display might look like this:

```
1350:0200  Periscope is a full-featured symbolic debugger, system monitor a
1350:0240  nd "break-out"..
1350:0250  switch for the IBM PC, XT, AT, and compatibles.
```

Examples:

'DA' displays memory starting where the last Display command left off.

'DA @FILENAME L20' displays memory starting at the symbol FILENAME for a length of 20H bytes.

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

Command: Display using Byte format

Syntax: DB [<range>]

Type: Internal

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 periods.

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

```
0000:0000  5E 03 3F 08 5D 0B F0 BF-62 0B F0 BF 67 0B F0 BF   ..?.].p?b.p?g.p?
0000:0010  ED 01 70 00 54 FF 00 F0-62 0B F0 BF 05 18 00 F0   m.p.T..pb.p?...p
```

Examples:

'DB' displays memory starting where the last Display command left off.

'DB @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>]

Type: Internal

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  083F:035E   BFF0:0B5D   BFF0:0B62   BFF0:0B67
0000:0010  0070:01ED   F000:FF54   BFF0:0B62   F000:1805
```

Examples:

'DD' displays memory starting where the last Display command left off.

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

'DD @VECTORLIST' displays memory starting at the symbol VECTORLIST.

Command: Display Effective address

Syntax: DE

Type: Internal

Description: This command is used to display the effective address of any reads or writes performed by the current instruction. It has no arguments.

The display shows the address of any reads or writes performed by the instruction at CS:IP. The display is always in byte format. This display mode is best used with a Data window -- the window will display the current effective address automatically before each instruction is executed.

If the current instruction reads memory, the effective address of the read is shown. If the instruction writes memory, the effective address of the write is shown. If the instruction reads and writes memory, only the read address is shown.

Examples:

If the current instruction is 'LODSB', the DE command displays memory in byte format starting at the read address, DS:SI.

If the current instruction is 'MOV [0123],AX', the DE command displays memory starting at DS:123H.

If the current instruction is 'MOVSW', the DE command displays memory starting at DS:SI but does not display the write address of ES:DI.

Command: Display using Integer format

Syntax: DI [<range>]

Type: Internal

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

This format is useful for examining data that is stored as an unsigned word integer. Each line of the display shows the starting segment and offset and up to 8 decimal numbers. The number displayed may be from zero to 65535. 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 'DI DS:SI L 20' the display might look like this:

```
15E6:1000      1      2      3      4  32764  32765  32766  32767
15E6:1010  32768  32769  32770  32771  65532  65533  65534  65535
```

Examples:

'DI' displays memory starting where the last Display command left off.

'DI DS:SI L 20' displays memory starting at DS:SI for a length of 20H bytes.

'DI @ARRAY' displays memory starting at the symbol ARRAY.

Command: Display using Number format

Syntax: DN [<range>]

Type: Internal

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

This format is useful for examining data that is stored as a signed word integer as used by BASIC and other languages. Each line of the display shows the starting segment and offset and up to 8 decimal numbers. The decimal numbers shown may vary from zero to 32767 (0H to 7FFFH) and from -32768 to -1 (8000H to FFFFH). 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 'DN DS:SI L 20' the display might look like this:

```
15E6:1000     +1     +2     +3     +4 +32764 +32765 +32766 +32767
15E6:1010 -32768 -32767 -32766 -32765     -4     -3     -2     -1
```

Examples:

'DN' displays memory starting where the last Display command left off.

'DN DS:SI L 20' displays memory starting at DS:SI for a length of 20H bytes.

'DN @ARRAY' displays memory starting at the symbol ARRAY.

Command: Display using Record format

Syntax: DR <address> <symbol>

Type: Internal

Description: This command is used to display a block of memory in an easy-to-read format using a previously-created record definition.

This format is useful for examining data that is part of a record, such as the PSP or a FCB. Each line of the display shows a field name and the data for the field in any format supported by Periscope. Any area of memory can be displayed using any record definition.

To use a record format, a record definition, or DEF file must exist. The program loader, RUN.COM, loads the record definitions from the DEF file. You can add record definitions using a text editor. See the sample file PS.DEF and the description of RS.COM in Chapter VIII. The following definition of the PSP is from the file PS.DEF.

```
\PSP            ; Program Segment Prefix
Int 20,b,2      ; DOS return
Top Mem,w,2     ; Amount of memory in paragraphs
Reserved,b,1    ; Reserved for DOS
Long Call,b,1   ; Long call to DOS function dispatcher
DOS Func,d,4    ; CS:IP of DOS function dispatcher
Terminate,d,4   ; CS:IP of DOS terminate address
Ctrl-Break,d,4  ; CS:IP of Ctrl-Break exit address
Error,d,4       ; CS:IP of critical error exit address
DOS Use,b,16    ; Reserved for DOS
Environ,w,2     ; DOS 2.00 Environment segment
DOS Use,b,2e    ; Reserved for DOS
PSP1,b,10       ; The first PSP read from the command line
PSP2,b,14       ; The second PSP read from the command line
```

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     5000
Reserved    00                                             .
Long Call   9A                                             .
DOS Func    F01D:FEF0
Terminate   0B42:012C
Ctrl-Break  0B42:0139
Error       0B42:0481
```

```
DOS Use   42 0B 01 01 01 00 02 FF FF FF FF FF FF FF FF FF   B...............
          FF FF FF FF FF FF                                 ......
Environ   125B
DOS Use   E2 FF 61 12 14 00 18 00 61 12 00 00 00 00 00 00   ................
          * 0001 LINES OF 00 SKIPPED *
          00 00 CD 21 CB 00 00 00 00 00 00 00 00 00         ..M!K..........
PSP1      00 20 20 20 20 20 20 20 20 20 20 20 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                                       ....
```

The syntax for this command is less flexible than that of the other Display commands. You must enter an address and a record name. The address should include a segment, since the address used for this command is kept separate from the address used for the other Display commands.

Examples:

Assume that the records PSP and FCB are defined (as in the file PS.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>]

Type: Internal

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 rather than as bytes. It reverses out the 'back words' style of storage used by the 8086 family. 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  035E  083F  0B5D  BFF0  0B62  BFF0  0B67  BFF0
0000:0010  01ED  0070  FF54  F000  0B62  BFF0  1805  F000
```

Examples:

'DW' displays memory starting where the last Display command left off.

'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>]

Type: External

Description: This command is used to modify memory.

The segment and offset must be specified for the address, to avoid accidental changes to 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 104H to 0, enter 0 when the byte is displayed. To back up to offset 102H, 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>

Type: External

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 '@' or a period. 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>

Type: Internal

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, to avoid accidental changes to 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>] [...]

Type: Internal

Description: The Go command is used to set sticky and 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 earlier. 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 break-out switch is pressed. The sticky breakpoints are remembered until cleared or PS.COM is rerun. If you have code and/or monitor breakpoints set and want to continue program execution without using any of the breakpoints, you can disable all breakpoints or use the QC command to exit Periscope.

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. Since the setting of a code breakpoint in the middle of an instruction can have bizarre results, set code breakpoints using symbol names where possible.

When a Go command is used, Periscope invisibly traces one instruction and then performs the Go. This allows you to set a breakpoint on the current instruction to repetitively go to the same address.

Examples:

All of the examples below invoke any sticky code breakpoints that are set and not disabled.

'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 Trace

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

Type: Internal

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

These breakpoints are Byte (BB), Interrupt (BI), Line (BL), Memory (BM), Port (BP), Register (BR), User (BU), 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 to 1000, but in many cases is the only way to find an elusive bug. Since this command is slow, try to use the normal Go command to get as close to the problem as possible.

The monitor breakpoints are remembered until cleared or PS.COM is rerun. If you have code and/or monitor breakpoints set and want to continue program execution without using any of the breakpoints, you can disable all breakpoints or use the QC command to exit Periscope.

Try to get in the habit of checking the breakpoint settings before using this command. Enter 'BA' to display the current breakpoints before entering 'GT'. This way you can make sure that the right breakpoints have been set.

If the external commands are not available, you will not be able to display or modify the breakpoint settings. If the external commands become unavailable due to a checksum failure, you should use RUN.COM to reload the external commands as soon as possible. If the /Z option was used when PS.COM was installed, you'll have to rerun it without the /Z option to make use of the sticky or monitor breakpoints.

For temporary and sticky code breakpoints, this command performs in the same fashion as the Go command described above. After a breakpoint, use the TB command to see the instructions preceding the instruction that caused the breakpoint.

Examples:

All of the examples below invoke the monitor and sticky code breakpoints that are both set and

enabled.

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

GT begins execution of the program with no temporary code breakpoints -- only sticky and monitor breakpoints that are enabled.

'GT ES:456' sets a temporary code breakpoint at ES:456.

Command: Hex arithmetic

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

Type: External

Description: This command is used to perform hexadecimal arithmetic.

Addition, subtraction, multiplication, and division are available. The standard operators 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>

Type: Internal

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

The port number may be from zero to FFFFH, although the IBM PC only supports ports from zero to 3FFH -- any larger number is effectively ANDed with 3FFH. 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

Type: Internal

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. 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 repeated instructions 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. If the current instruction is any form of a RET, IRET, or JMP (including conditional jumps) Periscope traces one instruction (to follow the code) instead of using a temporary code breakpoint.

There is one condition under which this command does not work. When you're tracing ROM no code breakpoints can be used, since you can't write to ROM.

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. Others include a CALL or INT that does not return control to the next instruction, and when a stack change is in progress (SS:SP points to an undefined area).

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' to trace to the next logical (not physical) instruction.

Command: Jump Line

Syntax: JL

Type: Internal

Description: This command is used to jump from one source code line to the next source code line.

This command is usable only when the current instruction corresponds to a high-level language source code line. The JL command sets a temporary code breakpoint on the next source code line in the same module. This is a quick method of moving through a high-level language program, keeping to the source code lines.

Example: Assume the current instruction is line 10 of the first source code module. Enter 'JL' to go to the next physical source code line number for the same module. Note: If your compiler does not generate line numbers for every line, the next line symbol may not be line 11.

Command: Jump Noswap

Syntax: JN

Type: Internal

Description: This command is used to jump to the next instruction without swapping screen displays.

This command is the same as the Jump command described earlier, except that it does not save and restore the screen display. On a single-monitor system, Periscope normally saves and restores the program's screen during each Jump instruction. This variant of the Jump instruction is provided so users of single-monitor systems can elect not to save the screen for instructions that do not change the program's screen. If this command is used for an instruction that modifies the screen, the screen output will be directed to Periscope's screen, possibly garbling it. If this occurs, use the K command to clear the screen or F10 to swap to the program's screen and back.

Examples:

See the examples for the Jump command.

Command: Klear

Syntax: K

Type: Internal

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>

Type: External

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, which equals 64K bytes.

To use this command, DOS must be available. 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>]

Type: External

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 be available. 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.

The LF 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. The LF 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>

Type: External

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, to avoid accidental changes to 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 original source block.

'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>

Type: External

Description: This command is used to enter data into the PSP for disk I/O and for naming files to be read or written by Periscope.

The name parameter is copied to a Periscope buffer for use with the Load and Write commands. It is then copied to the unformatted parameter area in the PSP, starting at CS:80H. 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 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.

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. If the PSP cannot be found, Periscope can still be used to read or write the file, presuming that DOS is available.

Since the Name, Load, View, Unassemble Source, and Write commands use DOS calls, Periscope must check to be sure that DOS is available. Hardware interrupts must be enabled and the vector for interrupt 21H must equal a value saved by PS.COM and RUN.COM.

The Name command also requires that the PSP's address has been set by RUN.COM and that the first four bytes of the PSP contain the bytes 'CD 20' followed by the top of memory size in paragraphs.

Examples:

'N C:COMMAND.COM' copies the file name to Periscope's internal file buffer and 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 file names to Periscope's internal file buffer and 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>

Type: Internal

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

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

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>]

Type: Internal

Description: This command is used to exit the debugger and display Periscope's quit options.

The optional sub-function is used to pre-answer the quit option 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. See the section in this chapter entitled 'Quit Options' for more information.

Examples:

'Q' exits the debugger and displays the quit options.

'QC' exits the debugger and continues execution of the interrupted program without setting any breakpoints.

'QS' exits the debugger and performs a short boot.

Command: Register

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

Type: Internal

Description: This command is used to display and 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 effective address and current instruction are shown in the U window if one is used.

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, line three would also 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. The effective address calculations and displays for the 8088, 8086, 80186, and 80286 real mode instructions are 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.

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 in 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 instruction references a byte value and the data byte is from 20H to 7FH, the ASCII equivalent of the byte is shown at the end of the line as a comment, in quotes. Illegal instructions are shown as '???'.

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. Ambiguous references are generated by a move of an offset to a register, such as '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 16-bit register names are

AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, and IP. The valid 8-bit register names are AH, AL, BH, BL, CH, CL, DH, and DL.

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 (=Ø)
Overflow	OV	NV
Direction	DN (STD)	UP (CLD)
Interrupt	EI (STI)	DI (CLI)
Sign	NG (negative)	PL (positive)
Zero	ZR (zero)	NZ (non-zero)
Auxiliary carry	AC	NA
Parity	PE (even)	PO (odd)
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' is a one-line method of changing 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. You can also enter 'R F ZR'.

Command: Register Restore

Syntax: RR

Type: Internal

Description: This command is used to restore the registers to a previously-saved state.

This command is usable only after a RS command has been used to save the registers. After a Restore has been performed, Restore is disabled until another Save has been performed.

Example: Assume the registers were previously saved using the RS command. Enter 'RR' to restore all registers to their values when the RS command was used.

Command: Register Save

Syntax: RS

Type: Internal

Description: This command is used to save the registers for later restoration.

The Register Save command saves the current state of the machine's registers and flags in case you need to restore the registers to that state at some later point. For example, assume you're debugging a subroutine. In many situations, it is very convenient to save the machine's registers and then start debugging the subroutine. If you discover a problem, you can then restart the subroutine by restoring the registers from their saved values.

To use the Register Save command, enter 'RS' when the Periscope prompt is displayed. Later, you can restore the registers to their saved state by using the RR command. This command does not restore any data areas modified by the subroutine.

To prevent accidental restoration of the registers, the RS command sets a flag that is cleared by the RR command. When this flag is cleared, the RR command generates Error 26, function not available.

Example: Enter 'RS' to save the machine registers. At any point later, the RR command may be used to restore the registers to their saved values.

Command: Search

Syntax: S <range> <list>

Type: External

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>

Type: External

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. If you're not using a symbol name for the address reference, be sure to specify the segment register that is to be used. For example, if you're searching for a procedure reference, specify CS.

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: Search for Unassembly match

Syntax: SU <range> <list>

Type: External

Description: This command is used to search memory for instructions that match a pattern.

This command can be thought of as a disassembly that only shows instructions that match a specified pattern. To use it, specify an address range that is to be searched and the pattern that is to be searched for. For example, to find all MOVSB instructions, enter MOVSB in quotes as the list argument.

Be sure to enter the search list as a quoted string in upper case letters. Note that there are eight spaces from the start of the mnemonic field to the start of the operand field -- to find all occurrences of 'MOV SP', enter 'MOV', five spaces, and 'SP'. Note: this command will not find source code lines or procedure labels -- just disassembled instructions.

Examples:

'SU CS:100 L 200 "MOVbbbbbSS"' searches from CS:100 for 200H bytes for any instructions that contain 'MOVbbbbbSS', where 'b' is a blank. There must be exactly five spaces after the 'MOV' for the command to work.

'SU @PSTART @PEND "POP"' searches from the address represented by PSTART through the address represented by PEND for POP instructions.

Command: Trace

Syntax: T [<number>]

Type: Internal

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: Trace Back

Syntax: TB * or F or L and [< +/- number>]

Type: Internal

Description: This command is used to display previously executed instructions.

The traceback buffer is used to save the machine registers each time Periscope is exited. This circular buffer can contain zero to 512 entries, depending on the PS.COM installation option '/B:nn'. Each entry contains the machine registers and an ascending sequence number. When displayed, the buffer shows the registers, sequence number, and a symbolic disassembly of the instruction indicated by the saved CS:IP. If PS.COM was installed with either '/B:0' or '/Z', this command is not available. The following example shows three consecutive entries.

```
AX=0000  BX=0000  CX=008B  DX=0000  SP=FFFE  BP=0000  SI=0000  DI=0000
DS=15E6  ES=15E6  SS=15E6  CS=15E6  IP=0137   NV UP EI PL ZR NA PE NC   #0001
                START:
15E6:0137  E81700       CALL    GETMEM

AX=0000  BX=0000  CX=008B  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=15E6  ES=15E6  SS=15E6  CS=15E6  IP=0137   NV UP EI PL ZR NA PE NC   #0002
                GETMEM:
15E6:0151  B106         MOV     CL,06

AX=0000  BX=0000  CX=0006  DX=0000  SP=FFFC  BP=0000  SI=0000  DI=0000
DS=15E6  ES=15E6  SS=15E6  CS=15E6  IP=0137   NV UP EI PL ZR NA PE NC   #0003
15E6:0153  BE0200       MOV     SI,0002
```

Use 'TB L' to display the last entry in the buffer and 'TB F' to display the first entry. The L or F may be followed by a number to display multiple entries. An optional '+' before the number displays entries in ascending sequence and a '-' before the number displays entries in descending sequence. The buffer and sequence number may be cleared by using 'TB *'.

Since Periscope adds to the buffer each time it is exited, watch out for possible discontinuities in the traceback buffer. If you're using the T or GT commands, there's no problem. If you're using the G or J commands, not all instructions will be 'seen' by Periscope -- they'll leave discontinuities in the traceback buffer. Note that the disassembly uses the current contents of memory at the saved CS:IP so

the disassembly may be incorrect if the instructions have been changed.

Examples:

'TB L3' shows the last three instructions traced, with the most recent first.

'TB F4' shows the first four instructions in the circular buffer, with the most recent last.

'TB -2' shows the two previous instructions, with the most recent first.

'TB 3' shows the next three instructions, with the most recent last.

'TB *' clears the traceback buffer and resets the sequence number to zero.

Command: Trace Noswap

Syntax: TN [<number>]

Type: Internal

Description: This command is used to execute the current program one instruction at a time without swapping screen displays.

This command is the same as the Trace command described earlier, except that it does not save and restore the screen display. On a single-monitor system, Periscope normally saves and restores the program's screen during each Trace instruction. This variant of the Trace instruction is provided so users of single-monitor systems can elect not to save the screen for instructions that do not change the program's screen. If this command is used for an instruction that modifies the screen, the screen output will be directed to Periscope's screen, possibly garbling it. If this occurs, use the K command to clear the screen or F10 to swap to the program's screen and back.

Examples:

See the examples for the Trace command above.

Command: Unassemble memory

Syntax: U [<range>]

Type: Internal

Description: This command is used to disassemble memory into the 8088, 8086, 80186, and 80286 real-mode instructions.

Memory is disassembled in either the ASM or Source/ASM mode as set by the UA and US commands respectively. ASM is the default mode.

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:

```
1261:0137 E81700        CALL    0151
1261:013A A13301        MOV     AX,[0133]
1261:013D BF1001        MOV     DI,0110
1261:0140 E82400        CALL    0167
1261:0143 A13501        MOV     AX,[0135]
1261:0146 BF2C01        MOV     DI,012C
1261:0149 E81B00        CALL    0167
1261:014C E83400        CALL    0183
1261:014F CD20          INT     20
```

With symbols:

```
                START:
1261:0137 E81700        CALL    GETMEM
1261:013A A13301        MOV     AX,[TOTMEM]
1261:013D BF1001        MOV     DI,0110                 ; TMEMORY
1261:0140 E82400        CALL    CONVERT
1261:0143 A13501        MOV     AX,[FREMEM]
1261:0146 BF2C01        MOV     DI,012C                 ; AMEMORY
1261:0149 E81B00        CALL    CONVERT
1261:014C E83400        CALL    DISPLAY
              DOSRET:
1261:014F CD20          INT     20
```

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 second line, address DS:0133 is referenced. When the symbol table is searched, the name TOTMEM is found and displayed instead of 0133.

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 'START' are found, since CS must be correct for the 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. Ambiguous references are generated by a move of an offset to a register, such as 'MOV DI,OFFSET TMEMORY'.

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 the Register command.

'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: Unassemble ASM instructions

Syntax: UA [<range>]

Type: Internal

Description: This command is used to disable source disassembly (the US command) and to disassemble memory into the 8088, 8086, 80186, and 80286 real-mode instructions.

Use this command to turn off source-level debugging of a high-level language. The U command described earlier disassembles memory in one of two modes -- ASM or Source. The ASM mode is the default and is set by the UA command. The source mode is set by the US command.

Examples:

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

'UA @A10 L 1' disassembles memory for one instruction starting at the symbol A10.

'UA' 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: Unassemble Source/ASM instructions

Syntax: US [<range>]

Type: Internal

Description: This command is used to enable source disassembly and to disassemble memory into the 8088, 8086, 80186, and 80286 real-mode instructions.

Use this command to turn on source-level debugging of a high-level language. The U command described earlier disassembles memory in one of two modes -- ASM or Source/ASM. The ASM mode is the default and is set by the UA command. The source mode is set by the US command.

When needed, Periscope prompts for the file name corresponding to the module being disassembled. Enter the file name and press return to display the high-level language source code along with the disassembled instructions. If the file is not found, the prompt is displayed again. If you press return without entering a file name, source disassembly is disabled. To change source files, enter 'UA', press return, and then enter 'US' again. Note that the View command turns off Source/ASM mode.

To display source code, the following conditions must be met:
- DOS must be available (see the description of the Name command for more information)
- A file buffer must be available (if PS.COM was installed with either '/E:0' or '/Z', this command is not available)
- Line symbols must be available for Periscope to be able to associate an instruction with a source code line
- Any disk I/O errors will cause an incomplete source display or none at all

Examples:

'US @A10' disassembles memory starting at the line symbol A10. The default length of 20H bytes is used.

'US' 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: View file

Syntax: V <name>

Type: External

Description: This command is used to view a source file from within Periscope.

The name is any legal file name, including drive, path, file, and extension. To use this command, DOS must be available. See the description of the Name command for more information. A file buffer must be available -- if PS.COM was installed with either '/E:0' or '/Z', this command is not available.

The file is displayed in the un-windowed area of the screen. Use the PgUp and PgDn keys to page up and down through the file. Use the up and down arrow keys to move up or down one line at a time. Use the Home and End keys to move to the start and end of the file. When you're finished viewing the file, press the Esc key to return to the debugger prompt. Note that Ctrl-Break cannot be used to terminate the view command -- Esc is the only way to exit View.

Examples:

'V C:PS.DEF' displays the file PS.DEF. Use the PgUp, PgDn, Up, Down, Home, and End keys to move through the file. When done, press Esc to return to Periscope's normal operation.

Command: Write Absolute disk sectors

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

Type: External

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, which is 64K bytes.

To use this command, DOS must be available. 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 then 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>]

Type: External

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 an address is not specified, CS:100 is used. To use this command, DOS must be available. 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. Do not attempt to write an EXE file that was not loaded with the LF command -- an EXE file loaded by RUN.COM is missing its header and cannot be written to disk.

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>

Type: External

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

Example: 'X 5051' displays
'5051h 20561d 0101 0000 0101 0001b PQ'.

Command: Xlate (translate) Address

Syntax: XA <address>

Type: External

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.

Example: 'XA 1234:5678' displays '179B8'.

Command: Xlate (translate) Decimal number

Syntax: XD <decimal number>

Type: External

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

The number must be from zero to 65535. The number may not have any punctuation, such as commas or periods. Numbers larger than 65535 can be translated, but the high order part is lost.

Example: 'XD 4660' displays
'5051h 20561d 0101 0000 0101 0001b PQ'.

Command: Option S

Syntax: /S

Type: External

Description: This command is used to make global changes to the values of segments 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 BASIC, FORTRAN, and Pascal.

Example: '/S F00D DS' changes the segment of all symbol table entries that are currently F00D 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.

Command: Option U

Syntax: /U <byte> [<address>]

Type: External

Description: This command is used to perform user-written code from Periscope.

To use this exit, a program similar to USEREXIT.ASM as described in Chapter VIII must be installed and PS.COM must be installed with the /I option. The number entered after the /U must be from nine to FFH. It is passed to the user-written program in register AH. Other information is passed, including the optional address entered on the command line. See Chapter VIII for more information.

USEREXIT.COM has a status display for the 8087 and 80287 numeric processors. To use USEREXIT, load it before PS.COM is loaded. Then, use '/I:60' when installing PS.COM. From within Periscope, enter '/U 87' to display the status of the numeric processor.

Example: Assuming that a user-written interrupt handler has been installed using INT 60H and that PS.COM had the '/I:60' installation option, '/U 9' performs user exit number 9.

Command: Option W

Syntax: /W D[<:byte>] R S[<:byte>] U[<:byte>]

Type: External

Description: This command is used to change Periscope's windows from within Periscope.

Periscope can window Data, Stack, Register, and/or Unassembly information. Once windows are established, the windowed data is displayed at a constant location on the screen and is updated after each command.

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 input line.

The D window shows data in any of the display formats. The address used initially defaults to 0:0, until a Display command is used. The window continues to show the same address until another display command is used. The output of the Display Record command is not shown in this window. Duplicate lines are not suppressed for windowed data. When RUN.COM is used to enter Periscope, the display address is set to DS:100.

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 disassembled instructions. The effective address of any memory reads or writes is shown in the first line 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. To page through memory, enter the U command with no address.

The byte parameter defines the length of the window in hex. 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 (16)
Register	2	2	2
Stack	2	1	10H (16)
Unasm	4	4	10H (16)

If you want to turn off all windowing, enter '/W' with no arguments.

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.

VII -- RUNning Your Program

The program RUN.COM is used to load COM or EXE files and enter the resident debugger. Periscope must be installed for RUN to work. For help, enter 'RUN ?' when the DOS prompt is displayed.

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 or Go command 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 path, 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 FCBs and command line in the PSP to look like the target program had been started directly from DOS.

RUN resets Periscope's display address to the PSP segment at offset 100H. It also clears some of Periscope's external tables, including the traceback buffer, source file buffer, screen buffers, record definition table, and symbol table. If any breakpoints are set, they are disabled to avoid possible interference with the new program. If the external code area has been garbled, RUN will reload it, using PS.COM.

➧ For Periscope II, RUN does not reload any garbled code.

If the file extension is COM or EXE, the specified directory is searched for a file of the form filename.DEF. If this file is found, it is presumed to be a record definition file. If it is not found, the file PS.DEF is used if available. The DEF file is then used to load Periscope's record and keyboard definition tables. If a DEF file is not found, the record definition table is cleared, but any keyboard definitions are not cleared. If an error is found in the DEF file, the record definition table will be partially loaded. See the description of RS.COM in Chapter VIII for more information.

If the file extension is COM or EXE, the specified directory is searched for a file of the form filename.PSS. If this file is found, it is used instead of the MAP file for the program's symbols.

If the PSS file is not found, the directory is searched for a file of the form filename.MAP. This file is then used to load Periscope's symbol table with address and line references. If a MAP file is not found, the symbol table is cleared. If an error is found in the MAP file, the symbol table is partially loaded. If you're using the Phoenix or DRI linkers, a PSS file must be used for symbols. See the description of TS.COM in Chapter VIII for more information.

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).

Starting with DOS 3.00, the drive, path, and filename of the loaded program is stored at the end of the environment space. Since RUN does not use the EXEC function to load programs, this area shows RUN.COM as the loaded program rather than the target program. The environment space is of variable length and is followed by DOS's memory allocation blocks, so it is not safe for anything but DOS to modify the environment. If your program uses this information, consider loading it normally and then loading symbols using SYMLOAD.COM (see Chapter VIII).

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, keyboard definition, and symbol tables are preserved.

Finally, control is passed to the resident portion of Periscope. When you've finished debugging your program, you can exit Periscope in one of three ways -- use a Go with no breakpoints set, Quit the debugger and then Continue execution, or Quit the debugger 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 have been reset to their original values.

VIII -- Using The Periscope Utilities

This chapter discusses these Periscope utility programs:

- PSPATCH.COM -- Used to patch PS.COM for not-quite compatible computers

- PUBLIC.COM -- Used to generate Public statements for assembly language programs

- RS.COM -- Used to verify and size record and key definitions

- SYMLOAD.COM -- Used to load Periscope's symbol tables from within your program

- TS.COM -- Used to verify and size MAP files and generate PSS files from the output of various linkers

- USEREXIT.ASM and USEREXIT.COM -- A sample program to perform user exits and user breakpoints from Periscope

Periscope supports DOS 2.00 pathnames for all file I/O. Programs that perform file I/O accept command lines containing any legal DOS pathname. Note that a pathname must be terminated by a carriage return (end of command line), a space, or a slash.

PSPATCH.COM

This program is used to patch PS.COM when Periscope is used on the less compatible 'compatibles'. It is not needed when Periscope is used on an IBM, Compaq, or other 99.44% compatible machine. It is needed when the entry points for BIOS interrupt vectors differ from the IBM standard.

To run this program, boot the system so that no memory-resident programs or device drivers are installed. Make sure that PS.COM is on the default drive and enter 'PSPATCH'. PS.COM is then patched as needed. If you get a message of the form 'Segment for interrupt xxH is not F000H -- Use /V option for this interrupt', be sure that these /V installation option(s) are used each time PS.COM is run. After patching PS.COM, install it and make sure that the Ctrl-Break keys, video display, keyboard, and parallel printer all function correctly.

PUBLIC.COM

This program is used to generate public statements for assembler programs. With Periscope, the more symbols you have, the better. To run this program, enter 'PUBLIC progname' from the DOS prompt. If no extension is specified, ASM is assumed. The program reads your source and writes a file of public statements to the file progname.PUB. You can then include or merge this file into your program.

For help, enter 'PUBLIC ?'. The output is in lower case unless you use the /U option after the file name. The program generates public statements for all data variables and procedures, subject to the rules below.

If the multi-line COMMENT statement is used, it must be the first word found in the source line. Nothing is generated for a name that starts with the numbers zero through nine. Any public statements generated for equates are absolute references and are not relocated in memory.

The source line is skipped if the first word found in the line is PUBLIC, EXTRN, END, ASSUME, ORG, INCLUDE, EVEN, NAME, TITLE, SUBTTL, PAGE, ELSE, WIDTH, %OUT, NOT, OR, AND, XOR, or MASK. A public statement is generated for the first word in a line if the first word ends in a colon or if the second word found in a line is DB, DD, DW, DQ, DT, PROC, LABEL, EQU, or =. STRUC definitions are skipped.

PUBLIC recognizes macros and conditional statements -- any items found inside these types of statements are ignored. Since these items can be nested, the program keeps track of the nesting level and generates public statements only when the nesting level is zero. The following items increment the nesting level -- MACRO, IFDEF, IFIDN, IFDIF, REPT, IRPC, IFNB, IRP, IFE, IFB, IF1, IF2, IF, and IFNDEF. ENDIF and ENDM decrement the nesting level.

RS.COM

This program is used to verify and size a record definition file. It reads a DEF file containing record and/or keyboard definitions and displays the number of definitions found and the total record table size required for the file. These definitions are loaded by RUN.COM to provide support for Periscope's DR command and for keyboard assignment using the Ctrl-Fn keys.

To run this program, enter 'RS progname' from the DOS prompt. The file extension is presumed to be DEF. The DEF file is presumed to be in the same format as the sample file PS.DEF. Use the size shown by RS.COM for the PS.COM /R installation option. For help enter 'RS ?'.

A section of the PS.DEF file is shown below.

```
\f1=k;dr cs:Ø .psp;
\f2=dr cs:5c .fcb;
\FCB            ; File Control Block
Drive,b,1       ; Drive Ø=default, 1=A, 2=B, etc.
File,b,8        ; File name
Extension,b,3   ; File extension
Block No,w,2    ; Current block number
Rec Size,w,2    ; Logical record size
File Size,d,4   ; File size
Date,w,2        ; Date of last update
Reserved,b,1Ø   ; Reserved for DOS
Rec No,b,1      ; Current relative record number
Rel Rec No,d,4  ; Relative record number from beginning of file
```

The first two lines of this file contain keyboard definitions for function keys F1 and F2. While in Periscope, these keyboard definitions may be called by pressing Ctrl and Fn at the same time. The keyboard definition must contain a back-slash, the function key number, an equal sign, and the desired keystrokes. No spaces are allowed until after the equal sign. If you want multiple commands, use a semi-colon to separate the commands. If you want the command to be executed immediately, place a semi-colon at the end of the line. No comments are allowed on keyboard definition lines. The maximum length of a keyboard definition's text is 64 characters for each of the ten function keys.

The third line of the file starts a record definition. 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.

Until another back-slash is found at the start of a line or the end of the file is reached, 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 may be any of the display formats, except E (effective address). 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, integer, or number 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 field line may be commented using a semi-colon preceding the comments.

SYMLOAD.COM

This program is used to load Periscope's symbol tables from within your program. This approach is needed when your program manages overlays or is not loaded by RUN.COM.

SYMLOAD is a memory-resident routine that is run once per DOS session (it can be rerun if needed). It attaches itself to an interrupt vector so your program can access it as desired. The default interrupt used by SYMLOAD is 67H, but this can be changed if needed. SYMLOAD uses DOS calls to read a symbol file, so DOS must be available for it to work.

To install SYMLOAD, enter 'SYMLOAD /I:nn', where nn is the interrupt number to be used to access SYMLOAD, when the DOS prompt is displayed. Be sure that Periscope has already been installed, since it is required for SYMLOAD to work. The '/I:nn' command-line entry is needed only when SYMLOAD is to use an interrupt other than 67H. If you do specify an interrupt, be sure that the interrupt is not already used by another program.

Once SYMLOAD has been installed, it may be accessed from your program by performing the appropriate interrupt. The registers used on entry are:

BX -- The value of your program's PSP segment. If your program is an EXE file, add 10H to the PSP segment. The symbols' segments will be relocated relative to the value passed in this register.

DS:DX -- Points to a PSS file name in ASCIIZ format. An extension of PSS is required. For example, to load 'C:SAMPLE.PSS', DS:DX would point to the string 'C:SAMPLE.PSS' followed by a binary zero. See the description of the program TS.COM for information on creating a PSS file.

On return, register AH contains the status of the operation. Register AL is used to return additional error information if a read error occurred. The possible values of AH are:

0 -- Successful symbol table load
1 -- Error reading PSS file (DOS error returned in AL)
2 -- Periscope symbol table too small for PSS file
3 -- Logical error in PSS file
4 -- Periscope is not installed

If the status returned is zero, the symbol table has been loaded successfully. Note that all addresses are relocated relative to the segment address passed in register BX, except for absolute references and for symbols whose segment was already in the range of F000H to FFFFH.

TS.COM

This program is used to verify and size a MAP file and optionally generate a Periscope symbol file. It 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 the file.

To run this program, enter 'TS progname' from the DOS prompt. The file extension is presumed to be MAP. The MAP file is presumed to be in the same format as the sample file SAMPLE.MAP. Use the size shown by TS.COM for the PS.COM /T installation option. For help, enter 'TS ?'.

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 C programs, variables defined outside the MAIN 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 /LI option at link time. Not all compilers support the line number option. The ones that are known to support this option are: Computer Innovations C, Lattice C, 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 first 26 modules are referred to as A through Z. Subsequent modules are referred to as AA through AZ, BA through BZ, etc.

If you have some symbols that you don't want to see, edit the MAP file and insert braces as desired to turn off symbol generation. A left brace ({) turns symbol generation off, and a right brace (}) turns it back on. Be careful when saving the MAP file -- don't let any TABs or high bits into the file.

Large MAP files are relatively slow to load, since RUN.COM must extract the symbols each time the corresponding program is executed. To reduce the load time for 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 supports Digital Research's LINK86 (version 1.3) and Phoenix's PLINK (version 1.4x). 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.

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. Line numbers are not available as symbols for LINK86.

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). This method can be used starting with PLINK 1.30, but does not support line numbers.

To get all possible symbols with PLINK, enter 'TS progname/Q/S'. The /Q option tells TS to use the EXE file for all symbols. The /S option tells TS to write the PSS file to disk, for later use by RUN. At link time, be sure to specify 'SYMTABLE' as a linker directive. This method can be used starting with PLINK 1.40.

USEREXIT.ASM and USEREXIT.COM

This sample program illustrates Periscope's ability to perform user-written code. User-written code can be used to perform breakpoint tests (see the BU command) and user exits (see the /U command).

The user-written code is installed as a memory resident program using an available interrupt from 60H to FFH. The program must be installed before PS.COM is run. Also, the PS.COM installation option /I:nn must be used, where nn is the interrupt vector used to access the user-written code. A signature of 'PS' must be present in the resident routine in the word preceding the interrupt entry point.

The registers used on entry are:

AH -- Contains the breakpoint test number of one to eight or the user exit number of nine to FFH.

AL -- Always zero.

DS:SI -- Point to Periscope's data area (see USEREXIT.ASM for the layout of the table). This

table contains the values of various variables used by Periscope. Any changes to the variables in this table are passed back to Periscope.

On return from a user breakpoint, register AL should be set to a binary one to indicate a hit. Any other value indicates that no breakpoint is to be taken.

On return from a user exit, register AL indicates whether the exit code has set a command to be executed by Periscope. If AL equals 2, Periscope reads the command line passed back from the user exit. The command line must start with a semi-colon and end with a carriage return. A user exit may use BIOS functions as desired. Periscope assumes the screen has been changed and moves the cursor to the bottom of the screen on return from a user exit.

Do not attempt to perform DOS functions from user-written code -- DOS may be active! You do not need to preserve the values of any registers other than SS and SP on return to Periscope. If your routine needs more than 32 words of stack space, switch to an internal stack, but be sure to switch back to the original stack before returning.

To install USEREXIT.COM, run the program from DOS. Then install PS.COM using the installation option '/I:60'. When Periscope is active, try using 'BU 1' and then 'GT' to get to a point where DOS is available. Try '/U 9' as an example of a user exit modifying the command line. If you have an 8087 or an 80287, try '/U 87' to display the numeric processor status.

IX -- Technical Notes

This chapter discusses miscellaneous technical topics:

- Debugging theory and limitations
- CPU differences
- DOS notes
- Debugging techniques
- Debugging device drivers and non-DOS programs
- Periscope internals
- The Submarine board
- The IBM Enhanced Graphics Adapter

Debugging Theory

The 8086 processor family provides two built-in functions that aid the debugging process. These are the breakpoint and single-step 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, Periscope is entered through a special entry point. The use of this entry point signals 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 modify the IP register to continue execution of the program being debugged.

Single-step is the other type of 8086 breakpoint. It is set by modifying the trap flag 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 monitor breakpoint commands (Byte, Interrupt, Line, Memory, Port, Register, User, and Word).

If an instruction outside Periscope clears the trap flag, it causes any tracing currently underway to be turned off. If an instruction external to Periscope sets the trap flag unexpectedly, Periscope ignores it.

Since Periscope cannot be used to trace itself, it is not possible for it to trace the execution of Interrupts 1, 2, or 3 or any other interrupts that point to Periscope's code areas.

CPU Differences

If an 8087 is used with interrupts enabled, an error will cause a NMI. Since Periscope uses the NMI, the debugger screen is displayed. Use the G command to continue execution. Since the 8087 may interrupt the 8088 at any point, CS:IP may contain any value. The 80287 does not use the NMI, so an error will not invoke Periscope.

Many PCs and XTs have an early version of the 8088 CPU that has a serious bug. This version can be identified by the copyright date of 1978 shown on the chip. The defect in these CPUs is that the instruction after an instruction that changes the stack segment is not protected from being interrupted. The defined method for changing the stack is to change the stack segment and then immediately change the stack pointer. If this process is interrupted, the stack may be in no man's land -- the beginning of a system hang. The fix is to get the later chip -- identified by copyright dates of 1978 and 1981.

On a correctly functioning 8088, any instruction that modifies any of the segment registers protects the next instruction from being interrupted. This is a bit of overkill, since changes to DS, ES, and CS do not need the protection that stack changes require. This is why you'll notice that Periscope skips instructions while tracing through an instruction that modifies a segment register -- the instruction was actually executed, but it was

invisible to Periscope. Be careful not to use Periscope's Jump or Go instructions to stop in the middle of a stack changeover, since this can cause the same problem as the defective 8088.

On the 80286, the instruction protection for changes to segment registers applies only to the stack segment -- instructions that change DS, ES, or CS do not protect the next instruction. Still, be careful not to use the Jump or Go instructions to stop in the middle of a stack change.

For the 80286, 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. If the code segment of these interrupts points to memory below PS.COM when it is installed, no change is made to the interrupt since it is already in use. Avoid use of hardware interrupt IRQ 5 (INT 0DH) on an AT for such things as a mouse, since a segment overrun that occurs when this interrupt does not point to Periscope will hang your system.

DOS Notes

If you're using DOS 2.00 or 2.10, you should be aware of bugs in DOS that can cause problems. The bugs involve improper changes to DOS's stack where the SP register is modified before the SS register. This can cause problems when the break-out switch is pressed at just the right time or when you attempt to trace through DOS. PC Tech Journal published the patch for DOS 2.10 in the November 1984 issue. The same principles apply to DOS 2.00, although the addresses are different. If possible, use a later version of DOS -- the bugs are fixed in DOS 3.00.

So that Periscope can perform file I/O safely, it checks the undocumented, but reliable, in-DOS flag. This byte contains zero if DOS is available. Periscope also checks to see that interrupts are enabled, to be sure that DOS was interrupted at a safe point. It verifies that the vector for INT 21H is the same as saved by PS.COM and RUN.COM. The location of the in-DOS flag can be found by performing INT 21H with AH=34H. ES:BX returns the address of the flag. If you want to perform file I/O and Periscope is telling you that DOS functions are not available, get CS:IP back to your code and try again. Do not attempt to modify the in-DOS flag or the interrupt flag in order to fool Periscope -- you can get a garbled disk directory very easily.

To make DOS available, you can use the user breakpoint in the sample program USEREXIT.ASM (See Chapter VIII for more information).

Debugging Techniques

While Periscope is active, the BIOS interrupt vectors it uses are reset to point to BIOS unless the '/V' installation option was used. To access some memory-resident programs while Periscope is active, you may have to use some of these options. For example, a program that displays the time may use interrupt 1CH. Unless you specify '/V:1C' when PS.COM is run, the clock program won't be active when Periscope is. Be aware that each '/V' option used reduces Periscope's dependability, since the interrupt vector is left pointing to RAM that can be corrupted.

When you press the break-out switch 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. If you know your program's Code Segment, enter 'BR CS EQ nnnn' to set a Register breakpoint when CS equals the desired value. If you aren't sure, use '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. If you're debugging a program that has line numbers as symbols, use the BL breakpoint to get back to your code.

To repetitively trace an instruction, enter the Go command once and then repeat it using F4. For example, if you want to watch the execution of line 110H, enter 'G 110' and press return. Then press F4 to repeat the go instruction as many times as desired.

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 a push of the button.

If you're programming in assembler, use the PUBLIC

program described in Chapter VIII to get symbolic access to as much of your program as possible. The more symbols you use, the easier it is to debug your programs.

If you're programming in C using a compiler by Computer Innovations, Lattice, or Microsoft, you can get debugging information such as line numbers and address references in your MAP file by using options provided with these compilers. By defining variables outside the MAIN, you can cause address references to be generated for program variables. At link time, be sure to specify a MAP file and the /LI and/or /M options as desired.

If you're programming in Microsoft BASIC, FORTRAN, or Pascal, 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 Display 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 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.

Debugging Device Drivers

After installing Periscope, press the break-out switch to get into the debugger. Then enter 'QS' to perform a short boot. This technique can be used to cross-boot into another operating system, a non-DOS environment such as a self-contained program, or back into DOS.

The short boot performs an INT 19H, and leaves NMI (INT 2) intact. If you are debugging non-DOS or pre-DOS programs such as device drivers, 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 15H and INT 5 remain unchanged across a short boot. If you've used the /J or /K options to allow entry to Periscope via Sys Req or Shift-PrtSc, these keys will still work after the short boot, presuming no change by another program.

Periscope uses RAM external to the Submarine board for screen buffers, record tables, symbol tables, key definitions, non-critical code, and the on-line help file. After a short boot, this memory is no longer an extension of DOS. If you need to keep Periscope from using any memory other than what is on the Submarine board, you may use the installation option '/Z'. This sets the size of the external buffers to zero. A screen buffer size of zero is also achieved by using the '/S:0' or '/A' installation options. The latter indicates that both monochrome and color displays are available. This method is much preferred over using '/S:0', since it preserves 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. The PS.COM '/L:nnnn' installation option can be used to specify the starting address for Periscope's tables.

➧ For Periscope II, all code and data areas are in normal RAM and the /Z installation option is not available. For debugging device drivers, use the '/L' installation option to place Periscope's code and data in the middle of memory.

Periscope 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.
- If INT 1 (single-step) is used to enter Periscope and a hardware interrupt is active, control is returned to the calling program.
- The write-protected memory on the Submarine board 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 breakpoint tests, control is returned to the program being debugged after reversing the above items.
- The temporary and sticky code breakpoints are reverted as needed.
- If a hardware interrupt is active, it is cleared.
- The speaker is turned off and the keyboard is turned on.
- 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 PS.COM was installed.
- Any needed screen saving/swapping is performed.
- Periscope's screen is displayed.

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. The screen is not restored if the JN or TN commands are used.
- BIOS interrupts are restored to their value on entry.
- The registers and stack are restored to their values on entry.
- The memory on the Submarine board is write-protected.
- If the DOS return is used, the speaker is turned off and the keyboard is turned on.

When Periscope is exited using the Short boot option, the following steps are performed:

- 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.

- User interrupts 60H through 67H are set to 0:0.
- BIOS interrupts are restored to their power-on values.
- The stack is relocated to the end of the first 64K of memory.
- The speaker is turned off and keyboard is turned on.

When Periscope is exited using the normal boot option, the following steps are performed.

- The stack is relocated to the end of the first 64K of memory.
- The speaker is turned off and keyboard is turned on.

The requirements for the proper operation of Periscope are:

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

- If trace or a monitor breakpoint is used, Interrupt 1 must point to the proper point in the resident portion of Periscope.

- If the break-out switch is used, Interrupt 2 must point to the proper point in the resident portion of Periscope. Also, NMI must not be disabled via an OUT of zero to port A0H on the PC and XT or an OUT of 1FH to port 70H on the AT.

- If a code breakpoint is used, Interrupt 3 must point to the proper point in the resident portion of Periscope.

The data fields used by Periscope are located at the beginning of the protected memory. The record definition PSDATA in the file PS.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'.

➧ For Periscope II, enter 'DR xxxx:100 @PSDATA', where xxxx is the starting segment for Periscope's tables.

The Submarine 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. See Chapter III for information on the switch settings.

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 break-out switch is pressed, a NMI is generated. Periscope detects that the switch was pressed by checking the high bit read from the second I/O port. If this bit is on, the switch was pressed, otherwise the switch was not pressed. To clear the switch, output any value to the second port. Clear the switch and NMI near the end of your program using the code shown below.

```
mov dx,301h             ; assumes Submarine port is 300h
out dx,al               ; clear the break-out switch - any value in al is ok
in al,61h               ; enable nmi
jmp short $+2           ; delay for 286
mov ah,30h              ; use 30h if pc or xt, 0ch if at
or al,ah
out 61h,al
jmp short $+2           ; delay for 286
not ah
and al,ah
out 61h,al
```

The IBM Enhanced Graphics Adapter

IBM's Enhanced Graphics Adapter (EGA) 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. Since the EGA has its own ROM BIOS for handling video functions (INT 10H), be sure to use a '/V:10' installation option when an EGA is present. This ensures that Periscope uses the EGA's BIOS rather than the standard BIOS.

Periscope supports the EGA's new video modes, including those using segments other than B000H and B800H. For single-monitor systems, a maximum of 32K of the screen buffer is saved and restored by Periscope.

If you plan to use the EGA's extended palette capabilities on a single-monitor system, be sure to use the extended save area (via SAVE_PTR as described in the EGA BIOS listing). This RAM data area is updated when the palette or overscan is set, allowing Periscope to restore the original palette when returning to your program. There are two problems with using this save area, however. First, the save/restore of the palette registers is incredibly slow. Second, the overscan or border color is not saved in a consistent location. If you use the 'set overscan' function (INT 10H, AH=10H, AL=1), the color is saved in a different location than if you use the 'set all palettes and overscan' function (INT 10H, AH=10H, AL=2). For use with Periscope, the latter call should be used. The best bet for using an EGA is to have a two-monitor system. If you use a single-monitor system, remember that the maximum screen size that can be saved and restored by Periscope is 32K.

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 69 -- transient or installation portion of Periscope (PS.COM)
70 through 89 -- RUN.COM
90 through 99 -- RS.COM
100 through 109 -- TS.COM
110 through 119 -- PSPATCH.COM
120 through 129 -- PUBLIC.COM
130 through 139 -- SYMLOAD.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 ?, A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, Q, R, S, T, U, V, 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 symbol (preceded by '@' or a period) or a one- to four-digit segment, a colon, and a one- to four-digit offset. A register name may be substituted for the segment or offset.

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

03 Missing segment -- Some commands that modify memory (Enter, Fill, and Move) require an explicit segment to reduce the chance of accidental memory modifications. 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, place a semi-colon 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 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.

If the Interrupt breakpoint (BI) or Line breakpoint (BL) command was used, this error indicates the absence of a plus or minus sign to enable or disable the breakpoint.

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

If the Port breakpoint (BP) command was used, this error indicates the absence of an input or output check. The valid operators are I and O 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. For some commands, the number is limited to two hex digits or the 8-bit registers. If part of a list, the number must be one or two digits and a register name cannot be used.

11 Invalid/missing register -- The register name must be AX, BX, CX, DX, SP, BP, SI, DI, DS, ES, SS, CS, or IP. The 8-bit registers AH, AL, BH, BL, CH, CL, DH, and DL may also be used. The register name may be in upper or lower case.

If this error occurs from the in-line assembler, it may mean that the register specified does not fit the instruction or is illegal (e.g., PUSH AL or POP CS).

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 breakpoints -- Too many addresses were input for the BC command (limit 16), the BB command (limit 8), the BP 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 Assemble command must be followed by a space or a U. The Breakpoint command must be followed by an A, B, C, I, L, M, P, R, U, or W. The Display command must be followed by a space, A, B, D, E, I, N, R, or W. The Enter command must be followed by a space or S. The Go command must be followed by a space or T. The Jump command must be followed by a space, L, or N. The Load and Write commands must be followed by A or F. The Register command must be followed by a space, F, R, S, or a register name. The Search command must be followed by a space, A, or U. The Trace command must be followed by a space, B, or N. The Unassemble command must be followed by a space, A, or S. The Xlate command must be followed by a space (same as the H sub-function), A, D, or H. The Option (/) command must be followed by a S, U, or W.

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/port less than first -- The second address or port number used with the BM or BP commands was less than the first number. Enter an address or port number greater than or equal to 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 '@' or a period 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 prefix (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 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 this error occurs, the table has a logical error, otherwise the table is full.

If the table is invalid, chances are good that it was garbled 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.

20 DOS functions not available -- DOS function calls are used by the Load, Name, View, Unassemble Source, and Write commands. Since DOS is not re-entrant, Periscope tests to be sure that DOS is available (not active). This is done by checking a flag set by DOS 2.00 and later versions. This flag must be zero, interrupts must be enabled, and the interrupt vector must equal its saved address for Periscope to allow DOS functions. If you receive this message, use the Go command to get back to your program's code and try the DOS function again. If you're in DOS, execute RUN.COM and try the DOS function again. For the LA and WA commands, Interrupt vectors 25 and 26 must contain the same address as when PS.COM or RUN.COM was run.

21 Not enough memory -- Insufficient memory is available to perform the Load command. Periscope

checks the amount of memory 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 Cannot open file -- Periscope was unable to open a file for input or output. If you're loading a file into memory, check the name as specified to the Name command. If you're writing a file, check that the filename is legal, the file is not a read-only file, and room exists in the directory for the file. This error can also occur if too many files are open.

24 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 installation option), or a carriage return. The total number of windowed lines, including a separator line for each window, must be 21 or less. The minimum size of the U window is four lines.

25 Read/write error -- A fatal error occurred when reading or writing a file or absolute sectors. Check the disk and filename and retry the command.

26 Function not available -- This error indicates that the desired command is not available. If the '/Z' installation option was used with PS.COM, the external commands are not available. These same commands are also not available if a checksum indicates that the external code area has been garbled. To reload a garbled external code area, execute RUN.COM. The internal commands are always available.

This error can also occur under several other conditions: when a RR command is used and no RS command has been previously used to save the registers; when a TB command is used and /B:0 or /Z was used when PS.COM was installed; when a US or V command is used and /E:0 or /Z was used when PS.COM was installed; or when a BU or /U command is used and no /I:nn was used when PS.COM was installed.

27 Unknown mnemonic -- An unknown mnemonic was specified to the in-line assembler. The assembler knows the mnemonics for the 8088, 8086, 80186, and 80286 processors. For the 80286, only the real-mode opcodes are supported. Check the mnemonic and try again. Segment overrides and other prefixes must be on a separate line preceding the instruction they affect.

28 B/W pointer missing -- An ambiguous instruction was specified to the in-line assembler. Certain instructions, such as 'MOV [SI],1', require a width indicator of byte or word. The instruction would be entered as 'MOV B [SI],1' or 'MOV W [SI],1', respectively. Note that Periscope's disassemble command shows the 'B' as 'byte ptr' and the 'W' as 'word ptr'.

29 Invalid memory reference -- An instruction that incorrectly references memory was specified to the in-line assembler. Check the register(s) and offset specified in the instruction to be sure that the memory reference is legal. For example, 'MOV AX,[DX]' is not legal, but 'MOV AX,[BX]' is legal.

30 Extra/missing argument(s) -- There are too many or too few arguments for the mnemonic specified. Check the number of arguments and try again. Note that the 80286 multiply immediate instruction must always be entered in the three-argument format.

31 Line symbol not found -- For the JL command, the current instruction (CS:IP) must be a line symbol. The next symbol found in the symbol table must also be a line symbol and the module prefix must match. Use the BL breakpoint to get to the next source code line and then use the JL command.

32 PSP not found -- The Name command was not able to locate the PSP. This error can be ignored if you need to read or write a file with the LF or WF commands. If you are trying to format the PSP, use RUN to re-enter Periscope.

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 /B, /C, /E, /I, /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 installation options.

42 Invalid installation option -- An unexpected entry was found in the installation options. The valid entries are: *, /A, /B:nn, /C:nn, /D, /E:nn, /F, /H, /I:nn, /J, /K, /L:nnnn, /M:nnnn, /P:nnnn, /R:nn, /S:nn, /T:nn, /V:nn, /W D:nn R S:nn U:nn, and /Z, 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.

➧ For Periscope II, installation options /M, /P, and /Z are not available.

43 Interrupt must be 08H, 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 chips on the board are properly seated in their sockets.

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!

49 DOS 2.00 or later required -- Periscope requires DOS 2.00 or later. Version 1.10 of Periscope is the last version that supports DOS 1.00 or 1.10.

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!

51 HELP file not found -- The help file, PSHELP.TXT was not found on the default directory. Change the

current directory, copy the help file to the current directory, or omit the /H option.

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

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 -- See the explanation of Error 24, above.

57 Unable to read response file -- An error occurred when PS.COM tried to read the response file. Check the file name and try again. Note that any installation options entered after the response file name are ignored. For example, 'PS /c:17 @c:std' sets the color attribute to 17H and then reads the rest of the options from the file 'c:std'. If you use 'PS @c:std /c:17', the color attribute is not used.

58 Backtrace buffer size must be from 0 to 10H (16) KB -- The size of the backtrace buffer specified with the /B option must be from zero to 10H K. Note that the number is in hex!

59 Invalid user interrupt vector -- The user interrupt vector specified with the /I option must be from 60H to FFH. The interrupt handler must be already installed using the specified interrupt. Periscope checks for the presence of the interrupt handler by reading memory at the interrupt's segment and offset. The word prior to the interrupt entry point must equal 'PS'. See the sample program, USEREXIT.ASM, for more information.

60 Load segment for Periscope tables must be from xxxx to yyyy -- The load segment as specified with the /L option must be greater than the current value

of the PSP plus 10H paragraphs. The load segment must also be less than the top of memory minus 1000H paragraphs. If the PSP is C00H and the top of memory is 5000H, then the allowable range for the load segment is C10H through 4000H.

61 Source buffer size must be from 0 to 10H (16) KB -- The size of the source buffer specified with the /E option must be from zero to 10H K. Note that the number is in hex!

62 Unable to write response file -- Periscope is unable to write the response file on the default drive. Check the disk and try again.

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

71 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.

72 Unable to read xxxxxxxxxxx -- An error occurred reading the indicated file. Check to be sure the disk is ready and that the file size shown by DIR indicates the true file size.

73 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.

74 Periscope (Version x.xx) not installed -- RUN cannot run without the corresponding version of Periscope installed. Install the correct version of Periscope and restart RUN.

75 Periscope not installed correctly -- RUN was unable to modify the protected memory. Reload Periscope and try again.

76 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.

77 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.

78 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 installed. If this is the case, restart Periscope with a larger symbol table, otherwise check the file and try again.

79 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.

80 Unable to read PS.COM -- RUN detected that the Periscope code outside the protected memory has been garbled. To restore this code, RUN must read PS.COM, which is assumed to be in the current directory. Make sure PS.COM is in the current directory and try again.

90 DEF file not found -- RS.COM 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 drive and file 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 description of RS.COM in Chapter VIII.

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.

94 DOS 2.00 or later required -- Periscope requires DOS 2.00 or later. Version 1.10 of Periscope is the last version that supports DOS 1.00 or 1.10.

100 MAP file not found -- TS.COM was not able to find a file of the specified name with an extension of MAP.

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

102 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.

103 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.

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

105 DOS 2.00 or later required -- Periscope requires DOS 2.00 or later. Version 1.10 of Periscope is the last version that supports DOS 1.00 or 1.10.

106 Unknown PLINK symtable type - x -- TS.COM encountered an unknown record type in the symbol table at the end of the PLINK file. Please notify us immediately if you encounter this error.

110 Unable to patch PS.COM (Version x.xx) -- An error occurred reading or writing PS.COM. Check the disk and file and try again.

111 Interrupt 1CH does not point to an IRET instruction -- PSPATCH found that interrupt 1CH does not point to an IRET instruction. Check to make sure that no device drivers or memory-resident programs are installed. If this does not clear up the problem, please call for assistance.

112 DOS 2.00 or later required -- Periscope requires DOS 2.00 or later. Version 1.10 of Periscope is the last version that supports DOS 1.00 or 1.10.

113 Wrong version of PS.COM -- The versions of

PSPATCH.COM and PS.COM do not agree. Use the same version of both programs and try again.

120 DOS 2.00 or later required -- Periscope requires DOS 2.00 or later. Version 1.10 of Periscope is the last version that supports DOS 1.00 or 1.10.

121 File not found -- The PUBLIC program was not able to find a file of the specified name. If no extension is specified, ASM is used. If an extension is specified, it is used.

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

123 Unable to read program file -- An error occurred reading the program file. Check the file and try again.

124 Unable to write .PUB file -- A disk error occurred when PUBLIC attempted to write the PUB file. Check the disk and try again.

130 Periscope (Version x.xx) not installed -- SYMLOAD cannot run without the corresponding version of Periscope installed. Install the correct version of Periscope and restart SYMLOAD.

131 Invalid command line input -- The only valid command line input to SYMLOAD is '/I:nn' where nn is the interrupt vector to be used by SYMLOAD. The hex number must be from zero to FF.

Index

Special Characters

A

B

C

D

E

F

G

H

I

J

K

L

M

T

U

V

W

X