

# Periscope<sup>®</sup>

***Professional Software and  
Hardware-Assisted Debuggers***

---

# Periscope Manual

## Version M50

**Software by** Brett Salter and Tony Christie

**Hardware by** Doug Kraul

**Manual by** Sharon Bailey

**Published by** The Periscope Company, Inc.

1197 Peachtree Street, Atlanta, GA 30361 USA

**Phone** 404/875-8080      **Sales** 800/722-7006

**FAX** 404/872-1973      **Support** 404/875-4726

Copyright © 1986-1990, The Periscope Company, Inc. (TPC). All rights reserved. No part of this publication, including the diskette, may be reproduced or distributed in any form or by any means without the prior written permission of the publisher. The software may be used by only one person on one computer system at a time. The software may be used by any number of people on any number of computer systems so long as there is no possibility that it is being used at more than one location at the same time.

We will attempt to fix errors you find in the software if you will provide us a way to recreate the problem on our computer systems. We welcome your comments and suggestions for improvements.

The entire risk as to the performance of this package is with the purchaser. TPC 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. TPC assumes no responsibility or liability of any kind for errors in the package or for the consequences of any such errors.

**Special thanks to Bob Smith and Glen Duff for their continuing help.**

Compaq is a registered trademark of Compaq Computer Corporation; IBM is a registered trademark and PC, XT, AT, and PS/2 are trademarks of International Business Machines Corporation; CodeView, Microsoft, MS and MS-DOS are registered trademarks of Microsoft Corporation; Periscope is a registered trademark of The Periscope Company, Inc.; other product and company names are trademarks of their respective holders.

---

---

# Table of Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
	REGISTRATION, UPDATES, AND UPGRADES . . . . .	2
	WARRANTIES . . . . .	3
	MONEY-BACK GUARANTEE . . . . .	4
	FCC COMPLIANCE . . . . .	5
	USING THE MANUAL . . . . .	5
	WHAT'S IN THE PERISCOPE PACKAGE . . . . .	7
	SYSTEM REQUIREMENTS . . . . .	8
<b>2</b>	<b>Getting Started</b>	<b>11</b>
<b>3</b>	<b>Tutorials</b>	<b>13</b>
	C TUTORIAL . . . . .	14
	ASSEMBLY TUTORIAL . . . . .	19
	MODEL IV TUTORIAL . . . . .	23
<b>4</b>	<b>Configuring Periscope</b>	<b>29</b>
	RUNNING SETUP AND CONFIG . . . . .	30
<b>5</b>	<b>Installing Model I Hardware</b>	<b>35</b>
	CHECKING FOR CONFLICTS WITH THE MODEL I, REV 3 BOARD . . . . .	36
	SETTING THE DIP SWITCHES ON THE MODEL I, REV 3 BOARD . . . . .	36
	POPULATING THE MODEL I, REV 3 BOARD . . . . .	39
	INSTALLING THE (MODEL I, REV 3) PLUS BOARD WITH MODEL IV . . . . .	40
	INSTALLING THE MODEL I, REV 3 BOARD . . . . .	40
	POPULATING THE MODEL I/MC BOARD . . . . .	43
	INSTALLING THE MODEL I/MC BOARD . . . . .	44
<b>6</b>	<b>Installing Model II Hardware</b>	<b>47</b>
	INSTALLING THE MODEL II BREAK-OUT SWITCH . . . . .	48

---

<b>7</b>	<b>Installing Model IV Hardware</b>	<b>53</b>
	CHECKING FOR CONFLICTS . . . . .	54
	SETTING THE DIP SWITCH . . . . .	54
	MODEL IV COMPONENTS . . . . .	55
	INSTALLING THE MODEL IV HARDWARE . . . . .	56
<b>8</b>	<b>Installing the Software</b>	<b>73</b>
	INSTALLATION OPTIONS . . . . .	74
	ALTERNATE START-UP METHODS . . . . .	86
<b>9</b>	<b>Using Periscope</b>	<b>89</b>
	SYMBOLS AND SOURCE SUPPORT . . . . .	90
	DEBUGGING AT THE SOURCE LEVEL . . . . .	93
	DEVICE DRIVERS . . . . .	94
	PLINK APPLICATIONS . . . . .	94
	.RTLINK APPLICATIONS . . . . .	94
	MICROSOFT WINDOWS APPLICATIONS . . . . .	95
	NON-DOS AND PRE-DOS PROGRAMS . . . . .	97
	DEBUGGING AT ROM-SCAN TIME . . . . .	97
	HARDWARE INTERRUPTS . . . . .	98
	MEMORY-RESIDENT PROGRAMS . . . . .	98
	USING AN ALTERNATE PC . . . . .	99
	USING AN EGA OR A VGA . . . . .	99
	PS/2 MACHINES . . . . .	99
	DEBUGGING SPAWNED (CHILD) PROCESSES . . . . .	100
<b>10</b>	<b>Periscope Utilities</b>	<b>101</b>
	CLEARING NMI (CLEARNMI) . . . . .	102
	YOUR PROGRAM'S INTERRUPTS (INT) . . . . .	102
	SETTING UP HOT KEYS (PSKEY) . . . . .	103
	USING AN ALTERNATE PC (PSTERM) . . . . .	104
	TESTING THE MODEL I BOARD (PSTEST) . . . . .	105
	TESTING THE MODEL IV BOARD (PS4TEST) . . . . .	106
	RECORD AND ALIAS DEFINITIONS (RS) . . . . .	107
	PERISCOPE'S PROGRAM LOADER (RUN) . . . . .	111
	IGNORING DOS MEMORY ACCESS (SKIP21) . . . . .	115
	LOADING SYMBOL TABLES (SYMLOAD) . . . . .	117
	DEBUGGING DEVICE DRIVERS (SYSLOAD) . . . . .	118
	GENERATING SYMBOL TABLES (TS) . . . . .	119
	CUSTOMIZING PERISCOPE (USEREXIT) . . . . .	123

---



WHEN DOS IS BUSY (WAITING) . . . . .	125
<b>11 Using Model IV</b>	<b>127</b>
MODEL IV CAPABILITIES . . . . .	128
MODEL IV COMPATIBILITY AND CAVEATS . . .	131
MODEL IV HARDWARE BREAKPOINT EXAMPLES . . . . .	133
MODEL IV IN PASSIVE REMOTE MODE . . . . .	134
<b>12 Keyboard Usage</b>	<b>137</b>
COMMAND EDITOR . . . . .	138
KEYBOARD ASSIGNMENTS . . . . .	138
<b>13 Periscope's Menus</b>	<b>145</b>
<b>14 Command Overview</b>	<b>149</b>
COMMAND SUMMARY . . . . .	150
BREAKPOINT COMMAND SUMMARY . . . . .	152
<b>15 Command Reference</b>	<b>155</b>

## **APPENDICES:**

<b>A Messages</b>	<b>281</b>
INFORMATIONAL MESSAGES AND PROMPTS . . .	282
ERROR MESSAGES . . . . .	284
<b>B Tech Support and Troubleshooting</b>	<b>305</b>
TECH SUPPORT . . . . .	306
TROUBLESHOOTING . . . . .	306
<b>C Command Parameters</b>	<b>311</b>
COMMAND PARAMETERS . . . . .	312
CHARACTER SEQUENCES . . . . .	317

## **INDEX**

---

---

# List of Figures

Figure	Page
4-1. Periscope Configuration Screen . . . . .	31
5-1. Setting Dip Switch SW1 on All Boards . . . . .	37
5-2. Setting Dip Switch SW2 on Model I, Rev 3 . . . . .	38
5-3. Layout of the Model I, Rev 3 Board . . . . .	40
6-1. Installation of Model II Break-out Switch . . . . .	49
7-1. Periscope Model IV . . . . .	56
7-2. 286 & 386 Chip Pullers . . . . .	57
7-3. Crowbar . . . . .	57
7-4. 80286 CPU Packages . . . . .	59
7-5. Cross-section: CPU/Flex Cable/Motherboard . . . . .	64
7-6. Layout of the 286/386 Pod . . . . .	69
7-7. 286/386 Pod: J4 & J5 Pin Assignment . . . . .	69
7-8. Layout of the 386 Rev 1 Pod . . . . .	70
7-9. 386 Pod, Rev 1: J5 Pin Assignment . . . . .	71
7-10. Layout of the Model IV Board . . . . .	72
8-1. Periscope Installation Screen . . . . .	87
10-1. A Section of the PS.DEF File . . . . .	108
14-1. Execution Speeds of Various Breakpoints . . . . .	153
15-1. Definition of the PSP from the File PS.DEF . . . . .	178
15-2. Sample Display Using the DR Command . . . . .	179
15-3. Data Access Types . . . . .	203
15-4. CPU Events for Various Access Widths . . . . .	204
15-5. Cycle Time by CPU Speed . . . . .	213
15-6. Hardware Trace Buffer in Raw Format . . . . .	214
15-7. Summary of Model IV Trace Buffer Commands . . . . .	217
15-8. Hardware Trace Buffer in Trace Format . . . . .	220
15-9. Hardware Trace Buffer in Unasm Format . . . . .	222
15-10. Sample Displays of Registers and Flags . . . . .	238
15-11. Display of RC Command . . . . .	242
15-12. Software Trace Buffer Display Using the TB Command	250
15-13. Software Trace Buffer Display Using the TR Command	250
15-14. Disassembly of FTOC Program in UA Mode . . . . .	253
15-15. Disassembly of FTOC Program in US Mode . . . . .	254
15-16. Disassembly of FTOC Program in UB Mode . . . . .	256
15-17. Periscope Window Lengths . . . . .	276
C-1. Flag Register Values/Mnemonics . . . . .	315

---

# Introduction



- Registration, Updates, and Upgrades
- Warranties
- Money-Back Guarantee
- FCC Compliance
- Using the Manual
- What's in the Periscope Package
- System Requirements

**W**elcome to the world of Periscope users! If you're anxious to get started on a bug hunt, go immediately to Chapter 2, Getting Started. Just be sure to read this chapter as soon as you can. The first four sections will familiarize you with important benefits of owning a Periscope and how to take advantage of those benefits. The last three sections will familiarize you with this manual, the components of your Periscope package, and the system requirements for running Periscope.

---

## REGISTRATION, UPDATES, AND UPGRADES

**Registration** . Please complete and return the registration card included with your Periscope package. When we receive your card, we will register your Periscope so that you can get free Tech Support when you call the support line during the specified hours or leave a message on BIX (see Appendix B). We no longer offer free first updates, but we will send you update notices, special offers on new products, and other pertinent information.


If you need Tech Support right away or if you think that the registration card may get lost in the mail, please send the card to us by overnight courier or by certified mail.

**NOTE:** We must have your registration card before we will provide Tech Support. The only exceptions is that if you purchased your Periscope directly from The Periscope Company (TPC), we will provide Tech Support for ten days from your order date without your registration card. We will also help with hardware installation questions, regardless of where you purchased your Periscope, without having received your card.

If you lose your registration card, you may still register. If you purchased your Periscope directly from TPC, there is no charge. If you purchased your Periscope from a dealer, you must provide us with proof of purchase (copy of invoice) and pay a registration fee (currently \$25). Once registered, you will receive all benefits of being a registered user.

**Hardware Updates.** We periodically revise the Periscope hardware to add functionality. For example, the current Periscope I, Rev 3 board can store up to one megabyte of RAM vs. the 56KB of RAM the previous Rev 2 board could store. The newer board therefore keeps all debugging information, including symbols, out of normal DOS memory. When there is such a revision in the hardware, we notify registered users of its availability and price, including any trade-in allowance being offered for the previous revision of the hardware.

---



**Upgrades.** We define an upgrade as a trade-in of one model of Periscope for a different, more expensive model of Periscope, for example when you trade in a Model II for a Model I. When you get a new version of Periscope software or a new revision of Periscope hardware without changing models, we call it an update rather than an upgrade. An example would be updating from software Version 4 to Version 5 or updating from a Model I, Rev 2 board to a Model I, Rev 3 board.

Registered users of Periscope may upgrade at any time. If you're trading in a model that has no board (II or II-X) for a model that does (I, I/MC, or IV), or if you've purchased any currently-produced model of Periscope within the last month, you can upgrade to a higher-priced model for the difference in price plus an upgrade fee (currently \$10). Current retail prices are used to calculate upgrade prices. Additional charges may apply if you're trading in software and/or hardware that's not current (Model I, Rev 2 board, Model III board, or Version 3.x software) or any model you purchased over a month ago. Call 800/722-7006 or 404/875-8080 for full details.

## **WARRANTIES**

TPC warrants all physical components of the Periscope package to be in good working order for a period of one year from the date of purchase as a new (or factory-refurbished) product. This warranty covers all hardware, i.e., boards, cables, switches, and adapters, plus the diskette, manual, binder, and quick-reference card.

Should any of these items fail to perform properly any time within the stated warranty period, TPC will, at its option, repair or replace it at no cost except as set forth in this warranty. Replacement parts or products will be furnished on an exchange basis only. Replaced parts and/or products become the property of TPC. No warranty is expressed or implied for damage caused by accident, abuse, misuse, natural or personal disaster, or unauthorized modification.

You may obtain warranty service by sending the defective

---

item to TPC during the warranty period. Call 404/875-4726 to get a return authorization number, write this number on the outside of the package, and include proof-of-purchase-date in the package you're returning. If you ship by mail or any common carrier, you must insure and accept all liability for loss or damage, prepay all shipping charges, and use a shipping container equivalent to the original packaging.

## **MONEY-BACK GUARANTEE**

If you purchase Periscope directly from The Periscope Company and the package does not perform to your satisfaction, you may return the complete, undamaged package to us within 30 days of our shipping (invoice) date, and we will refund your purchase price less shipping charges. On returns of Model IV, we may also deduct a restocking fee of up to 15% of your purchase price. (We may, depending on the circumstances and solely at our option, waive the restocking fee.)

Here's the procedure you must follow to return a Periscope package under this guarantee:

- Call 404/875-8080 for a Return Authorization (RA) number.
- Mark the RA number clearly on the outside of the package being returned. If no RA number is on the package, we may refuse it and you'll have to reship it at your expense.
- Return the package so that we receive it within the 30-day guarantee period.

**NOTE:** If we do not receive the package within 30 days of our original invoice (shipping) date, we may refuse it. The money-back guarantee is not valid after 30 days.

- Be sure that all items are in the package, including the manual, binder, diskette, quick-reference card, registration card (if you haven't already sent it to us), and all hardware for the model you're returning. If any items are missing, a charge for the missing item(s) will be deducted from your refund.

- 
- Be sure that the package is packed so that it is not damaged in shipping, and use a reliable carrier. If any items are damaged when we receive the package, a charge for the damaged item(s) will be deducted from your refund.

## FCC COMPLIANCE

**Warning:** This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instruction manual, may cause interference to radio communications. It has been tested and found to comply with the limits for a Class A computing device pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference when operated in a commercial environment. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

## USING THE MANUAL

**Notes.** We point out Periscope model differences, warnings, exceptions, and special instructions with a note like this:

**NOTE:** Here's how we make sure you don't miss this information!

**Windows.** When discussing 'windows', we specify Microsoft and capitalize the 'w' in Windows when we mean the Microsoft Windows environment. Whenever we use the generic term 'windows', with no capital 'w', we mean Periscope's windows.

**Typography.** The normal font used in the body of the manual is a proportional font, meaning different spacing is used for different characters to make the text appear evenly-spaced and more readable. This can make it difficult for you to determine the spacing of items you might enter into the computer. Therefore, when we specify syntax, a command or options that you might enter, messages that appear

---

on the screen, etc., we use a fixed-space font, which uses exactly the same amount of space for all characters. This sentence is formatted in the normal proportional font used for most body copy. This sentence is formatted in the fixed-space font that is used for "entry" items, screen messages, etc.

A ♦ (diamond) appears at the end of each chapter to indicate the end of the chapter.

**Models Covered.** Use this manual with the models of Periscope described below. They all include very similar software, but different hardware.

**Periscope I, Rev 3** includes a memory board with 512K of write-protected RAM and a remote break-out switch. If you purchase the version with no write-protected RAM, you must add either 512K or one megabyte of RAM to the board to run it. The RAM protects Periscope's software and all debugging information from programs that overwrite low memory, where debugger software normally resides. Also, since Periscope I uses no memory in the lower 640K of your system, the debugger doesn't compete with your program for space in normal DOS memory.

The optional remote break-out switch that plugs into the phono jack on the board's mounting bracket enables you to stop an executing program and enter Periscope. This break-out capability lets you 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.

**NOTE:** The Model I, Rev 3 board is also referred to as the "Plus board" when used in conjunction with a Model III or Model IV.

**Periscope I/MC** is the version of Periscope I for PS/2s and other Micro Channel bus machines. It is functionally identical to Periscope I described above, except that the board will run with up to two megabytes of RAM.

**NOTE:** When we refer to "Model I" or "Periscope I", we mean both Model I, Rev 3 for PCs and Model



---

I/MC. Otherwise, we will specify which Model I we mean.

**Periscope II** includes a remote break-out switch, but no write-protected memory board. The switch taps into an expansion slot that is already in use, so you do not need a dedicated slot. The Model II break-out switch performs the same function described above for the Model I break-out switch.

**Periscope II-X** is software only, with no break-out switch or board. It does not support Interrupt 2 (NMI), as do all other models of Periscope. Because Model II-X works on machines that use NMI for their own purposes, it may work in environments where there's a conflict with the other models of Periscope.

**Periscope III.** Since we no longer produce Periscope III hardware, we now document its installation and special hardware capabilities in a separate addendum. We do still support Model III and will continue to do so indefinitely, so we do reference it in this manual where appropriate.

**Periscope IV** includes a remote break-out switch, a board with a real-time trace buffer and hardware breakpoints, a pod, and if the pod is the 80286/80386 pod, a flexible printed circuit cable for an 80286 and/or an 80386 machine. The break-out capability is the same as for Model I. The Model IV board monitors the system CPU, and with your program executing at full speed, captures execution information and/or stops at specified hardware breakpoints. Model IV supports 80286 and 80386 machines running up to 25MHz and supports remote debugging.

## WHAT'S IN THE PERISCOPE PACKAGE

If you discover that any of the items described below for your model of Periscope are missing from your package when you receive it, please contact your dealer immediately.

### All models include these items:

- Diskette with Periscope software (5.25" or 3.5")
- Manual with binder

- 
- Quick-reference card
  - Registration card

**Plus these model-specific items:**

- **Periscope I:**  
Remote break-out switch  
Model I, Rev 3 board with either zeroK or 512K RAM
- **Periscope I/MC:**  
Remote break-out switch  
Model I/MC board with 512K RAM
- **Periscope II:**  
Remote break-out switch
- **Periscope II-X:**  
No additional items
- **Periscope IV:**  
Remote break-out switch  
Model IV board
- **Model IV Options:**  
This is continually changing. Please see the file  
NOTES.TXT for the current information.

**NOTE:** You'll find a description of the files on the distribution diskette in the file NOTES.TXT.

## **SYSTEM REQUIREMENTS**

**The Periscope software requires:**

- an IBM PC, XT, AT, or PS/2; Compaq, including the 80386 and the 80486; or other fully software-compatible personal computer
- PC/MS-DOS 3.00 or later
- one disk drive
- an 80-column monitor
- zeroK to ~ 80K available RAM (up to 140K when using menus)

**The Periscope hardware requires:**

- The **Periscope I** board requires an IBM PC-compatible (ISA or EISA) bus, one full-length slot, and 32K of address space above the lower 640K but in the first megabyte of system memory.
- The **Periscope I/MC** board requires an IBM PS/2-com-

---

patible (MCA) bus, one full-length slot, and 32K of address space at C000:0, C800:0, D000:0, or D800:0.

- The **Periscope II** break-out switch requires an IBM PC-compatible (non-MCA) bus. It does not require its own slot, however, since there's no board.
- The **Periscope IV** hardware requires a machine with an 80286 or 80386 CPU running up to 25MHz with zero or more wait states. (80286 machines with LCC or PLCC CPU packages require an adapter.) The board requires an ISA or EISA bus and one full-length slot. The optional Plus board is the Periscope I board (see requirements above).

Please call Tech Support if you have compatibility questions or problems. ♦



# Getting Started



## ■ Overview

**T**his chapter outlines the steps you need to follow to start debugging with Periscope. It also directs you to the chapters in the manual where you'll find the information you need to perform each step.

---

## OVERVIEW

### Step 1 — Install the hardware.

- Model I users, see Chapter 5.
- Model II users, see Chapter 6.
- Model IV users, see Chapter 7.
- Model III users, see the Model III Addendum.

### Step 2 — Configure Periscope.

- See Chapter 4.

### Step 3 — Take the tutorials.

- See Chapter 3.

### Step 4 — Install the software.

- See Chapter 8.

### Step 5 — Begin using Periscope.

- See Chapter 9 for information on using Periscope in various environments.
- See Chapter 10 for details on using the Periscope utility programs.
- Model IV users, see Chapter 11 for an overview and examples of using Model IV's hardware capabilities.
- For reference see:
  - Chapter 12 for keyboard usage.
  - Chapter 13 for menus.
  - Chapter 14 for a command overview.
  - Chapter 15 for command details.
  - Appendix A for informational and error messages.
  - Appendix B for common technical questions and troubleshooting hints,
  - The NOTES.TXT file for other technical details. ♦

# Tutorials

- **C Tutorial**
- **Assembly Tutorial**
- **Model IV Tutorial**

**W**e intend that the following tutorials help you get familiar enough with Periscope that you can begin using it to debug your own programs right away. The C tutorial focuses on symbolic and source-level debugging for high-level languages. The assembly tutorial is more general. We recommend you take both tutorials regardless of the programming language you use. The Model IV tutorial focuses on using Model IV's hardware-assisted debugging capabilities. We strongly recommend that all Model IV users take this tutorial before beginning to use Model IV. Because the tutorials do not cover all commands, we also recommend that you supplement what you learn taking the tutorials with information contained elsewhere in the manual. Chapter 12 covers keyboard usage; Chapter 13 covers the Periscope menu system; Chapter 14 provides an overview of commands; and Chapter 15 covers all commands.



---

**NOTE:** Don't skip, add, or change any steps in any of the tutorials, since any step may depend on previous steps.

## C TUTORIAL

This tutorial takes you through a guided tour of Periscope from the high-level language perspective. It uses the program FTOC, with these files:

**FTOC.C** is the source code for FTOC. Periscope uses this file to display source code.

**FTOC.DEF** is the optional FTOC definition file, which contains two alias definitions and one record definition. Periscope uses it to read alias and record definitions into memory. You create a DEF file with an editor as a standard ASCII text file. See the description of the utility program RS in Chapter 10 for more information about DEF files.

**FTOC.EXE** is the executable program.

**FTOC.MAP** is the MAP file produced by the linker. Unless you use the TS program to create symbols from another source, Periscope uses the information in the MAP file to replace memory addresses with symbols from the program you're debugging. You create a MAP file at link time by specifying a MAP file and the link options `/LI` and `/M`. This file contains public and line-number symbols. In this tutorial we'll use symbols stored at the end of the FTOC.EXE file by the link option `/CO`, instead of those in the MAP file, as explained in Step 3 below.

### Step 1 — Configure Periscope for your system.

Run **SETUP** and **CONFIG** per the instructions in Chapter 4 to configure the Periscope software for your system. For purposes of this tutorial at least, you should enable Periscope's menu system and select the CodeView function-key emulation when you run **CONFIG**.

All Periscope distribution disks contain the same files,



*Duplicate page - Should be pages 19+20*  
*(Missing)*

regardless of model. SETUP copies the files from the distribution disk to your working directory. CONFIG generates the executable Periscope software.

## **Step 2 — Install the Periscope software.**

Make the Periscope directory the default directory, then enter `PS /H /T:4` from the DOS prompt. This loads Periscope into memory (`PS`), loads the on-line help and interrupt comment files into memory (`/H`), and allocates 4KB for symbol tables (`/T:4`).

**NOTE:** If you have a Model I board in your system, enter `PS`. You do not need the options because Model I defaults to 128K for symbols and to loading the help file. If you changed the DIP switches on the board, however, do enter the memory and/or port options.

## **Step 3 — Create a Periscope symbol file with local symbols.**

We used Microsoft C to compile FTOC. Other compilers may use different names for compiler-generated symbols.

Microsoft LINK does not place local symbol information in the MAP file. To provide you with access to local symbols as well as public and line-number symbols while you're debugging, Periscope must go to the EXE file for symbolic information. Enter `TS FTOC /E` to create a Periscope symbol file from FTOC.EXE. (To use the MAP file for symbols, you would enter `TS FTOC`.)

## **Step 4 — Use Periscope's program loader to run FTOC.**

Enter `RUN FTOC` and press return. RUN displays informational messages and the address of the PSP, then switches to Periscope's screen. Normally, Periscope starts at the very beginning of the program and displays assembly code, but we've included a special alias in the DEF file, `\X0=G_MAIN`, that causes Periscope to immediately go to `_MAIN`, which is the beginning of the C source code.

## **Step 5 — Display Periscope's windows.**

---

Since we didn't specify any windows when we installed Periscope in Step 2, you see Periscope's default windows:

- The first line of the screen contains the menu bar which now shows the function key assignments for CodeView emulation. If you don't see this, start over at Step 1!
- The next two lines of the screen show a data window which can be used to display memory in various formats.
- The next window is the watch window, which can be used to monitor various memory and I/O port locations. The only watch item currently set shows the Periscope software version and a suffix that indicates the Model of Periscope you're using.
- The next window is the disassembly window. This is used to display the program being debugged in source, assembly, or mixed mode.
- On the right-hand side of the screen, there are two vertical windows for the system registers and the stack.

The current stack pointer is at the top of the stack window. (A chevron points to the value of the BP register when BP is in the range shown by the stack.) To toggle the register window off and on, press the Alt-R keys. To toggle the stack window off and on, press the Alt-S keys. You can specify your own windows using either the `/W` installation option or the `/W` command.

**NOTE:** When we refer to an installation option, we mean an option that is used when PS.COM is loaded into memory. See Chapter 8 for the various installation options available with Periscope. When we refer to a command, we mean a command that is used while Periscope's screen is displayed. See Chapter 15 for the various commands available with Periscope.

## **Step 6 — Activate the menu system.**

Press Alt-M to get into the menu system. Notice how the menu bar changes from showing the key prompts to showing the various menus available. Use the right arrow to

---

regardless of model. SETUP copies the files from the distribution disk to your working directory. CONFIG generates the executable Periscope software.

## **Step 2 — Install the Periscope software.**

Make the Periscope directory the default directory, then enter `PS /H /T:4` from the DOS prompt. This loads Periscope into memory (`PS`), loads the on-line help and interrupt comment files into memory (`/H`), and allocates 4KB for symbol tables (`/T:4`).

**NOTE:** If you have a Model I board in your system, enter `PS`. You do not need the options because Model I defaults to 128K for symbols and to loading the help file. If you changed the DIP switches on the board, however, do enter the memory and/or port options.

## **Step 3 — Create a Periscope symbol file with local symbols.**

We used Microsoft C to compile FTOC. Other compilers may use different names for compiler-generated symbols.

Microsoft LINK does not place local symbol information in the MAP file. To provide you with access to local symbols as well as public and line-number symbols while you're debugging, Periscope must go to the EXE file for symbolic information. Enter `TS FTOC /E` to create a Periscope symbol file from FTOC.EXE. (To use the MAP file for symbols, you would enter `TS FTOC`.)

## **Step 4 — Use Periscope's program loader to run FTOC.**

Enter `RUN FTOC` and press return. RUN displays informational messages and the address of the PSP, then switches to Periscope's screen. Normally, Periscope starts at the very beginning of the program and displays assembly code, but we've included a special alias in the DEF file, `\x0=G_MAIN`, that causes Periscope to immediately go to `_MAIN`, which is the beginning of the C source code.

## **Step 5 — Display Periscope's windows.**

---

Since we didn't specify any windows when we installed Periscope in Step 2, you see Periscope's default windows:

- The first line of the screen contains the menu bar which now shows the function key assignments for CodeView emulation. If you don't see this, start over at Step 1!
- The next two lines of the screen show a data window which can be used to display memory in various formats.
- The next window is the watch window, which can be used to monitor various memory and I/O port locations. The only watch item currently set shows the Periscope software version and a suffix that indicates the Model of Periscope you're using.
- The next window is the disassembly window. This is used to display the program being debugged in source, assembly, or mixed mode.
- On the right-hand side of the screen, there are two vertical windows for the system registers and the stack.

The current stack pointer is at the top of the stack window. (A chevron points to the value of the BP register when BP is in the range shown by the stack.) To toggle the register window off and on, press the Alt-R keys. To toggle the stack window off and on, press the Alt-S keys. You can specify your own windows using either the /W installation option or the /W command.

**NOTE:** When we refer to an installation option, we mean an option that is used when PS.COM is loaded into memory. See Chapter 8 for the various installation options available with Periscope. When we refer to a command, we mean a command that is used while Periscope's screen is displayed. See Chapter 15 for the various commands available with Periscope.

## **Step 6 – Activate the menu system.**

Press Alt-M to get into the menu system. Notice how the menu bar changes from showing the key prompts to showing the various menus available. Use the right arrow to

---

browse through the various menus. The menus are a convenient way of executing infrequently used or complicated commands, but because of the extra keystrokes required, you'll want to either enter the most frequently used commands directly or set them up on function keys as has been done with the CodeView function key emulation.

Press Esc now to return to the command interface.

### **Step 7 — Display the three different disassembly modes.**

You can see FTOC in the disassembly window in three modes : UA shows assembly only; UB shows both source and assembly; and US shows source-only. (The source-only display shows assembly code until the first source line reference is found, then displays source-only.) Enter UA, then US, and then UB to see the differences in the displays. Periscope defaults to US mode, unless it finds no line numbers in the symbol file.

### **Step 8 — Use Periscope's help feature.**

Enter ? and press return. You'll see a summary of all commands. Now enter ? D to get help on the Display command.

### **Step 9 — Set up local variables in the watch window for monitoring.**

Enter G #10 to go to line 10 in FTOC. Enter W I STEP to display STEP in integer format. Enter W S FAHR to display FAHR in short-real format. Neither variable is initialized yet, so their values mean nothing.

### **Step 10 — Trace through the program at the source-line level, then at the assembly level.**

Enter JL or press F10 to step to the next source line in the program. The Jump Line command uses the Jump command to step through your program until the 'current' instruction is the beginning of a source line.

The Jump command steps to the next sequential assembly level instruction. The Trace command is the same as the

---

Jump command, except that it traces into called routines and interrupts, instead of staying at the instruction level. Enter J and T a few times and note how the disassembly window changes.

**Step 11 – Go to the next call of the library procedure `_PRINTF`.**

Enter G `_PRINTF`. At this point, you're away from FTOC's source code, so you see no source lines. Also, you don't see the names of the local variables in the watch window, since the procedure in which they are defined is not currently executing.

**Step 12 – Analyze the stack for return points.**

Enter SR. The first item, FTOC#17+0020, indicates the instruction after the call to `_PRINTF`. To quickly return to the mainline code, enter GR, then enter JL or press F10 to get to line 18.

**Step 13 – Use a more general, but very slow, method of getting to the next source-code line.**

Enter G `_PRINTF` again, then enter TL or press F8. This command traces instructions until the next source line, line 18, is executed. Check the value of FAHR in the watch window versus the value just displayed by FTOC. (Press Alt-O or F4 to switch back to the program's display.)

**Step 14 – Monitor the value of the local variable FAHR.**

Enter G FTOC#18 or press F7 a few times to go to the next several executions of line 18. Notice how the value of FAHR in the watch window changes each time. You can use the short form of the line number, G #18, when you are already 'in' the module.

**Step 15 – Display local symbols using a record definition.**

Enter /w to turn the windows off temporarily. Enter // and press Alt-I to display the local symbols. Next, press Ctrl-F1 to call a keyboard macro that executes the command

---

### Step 6 — Use Periscope's help feature.

Enter `?` and press return. You'll see a summary of all commands. Now enter `? D` to get help on the Display command.

### Step 7 — Display the PSP in the data window.

Enter `DB CS:0`. You'll see the first 32 bytes of the PSP in Byte format. To see a more useful display, enter `/W` to turn Periscope's windows off. Then enter `DR CS:0 PSP`. You'll now see the PSP displayed in Record definition format, which 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 in Chapter 10). To see the record definitions available, press Alt-E before entering anything after the Periscope prompt. To restore Periscope's windows, press Ctrl-F9.

### Step 8 — Set up two variables in the watch window for monitoring.

Enter `W I TOTMEM`, then `W I FREMEM` so you can see the values of TOTMEM and FREMEM in integer format.

### Step 9 — Practice setting and executing breakpoints.

To set a sticky code breakpoint at the end of the program (DOSRET), enter `BC DOSRET` and press return. (You can see the available symbols by pressing Alt-I.)

Now, set a word breakpoint on the value of the variable TOTMEM changing from zero to any other value. Enter `BW TOTMEM NE 0` and press return. To display the current breakpoints, enter `BA ?`. You'll see `BC DOSRET` and `BW TOTMEM NE 0000`.

Enter `GT` to execute SAMPLE with both of the breakpoints activated. Execution will stop at the instruction after the one that moves register AX into TOTMEM. Press the Pad-Minus key twice to move the disassembly window back two lines.

---

To see the instruction that caused the breakpoint, enter **TB**. Periscope's traceback command shows previously-executed instructions in a full-screen mode using the software trace buffer. Use **PgUp** and **PgDn** to scroll through the buffer. Press the **Esc** key to display Periscope's prompt.

Now, enter **BW \*** to clear the word breakpoint. Enter **BA** (or **BA ?** as you entered earlier) to display the breakpoints. Only one breakpoint is still set, **BC DOSRET**.

#### **Step 10 — Convert hex to decimal.**

The watch window shows the current value of **TOTMEM** as a decimal value. To display the value in hex, enter **W W TOTMEM**. 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.

#### **Step 11 — Trace through the program at the assembly level.**

Use the **Jump** command to step through the next few instructions. Enter **J** and press return. Then press **Alt-D** to repeat the command. When you get to the **RET** instruction, use the **J** command one more time to get back to the main-line code.

#### **Step 12 — Disassemble and verify the number conversion routine.**

Enter **U CONVERT** to disassemble the number conversion routine. To restore the disassembly to the current instruction, enter **R** and press return.

To check the number conversion routine, **CONVERT**, enter **G #35** three times to get to the instruction after the first call to **CONVERT**. Enter **DA TOTAL** or **DA TMEMORY** to display the converted value. This value should agree with the value of **TOTMEM** in the watch window.

#### **Step 13 — Return to DOS.**



---

The sticky code breakpoint for DOSRET is still in effect, so enter **G** to go to DOSRET, then enter **G** again to return to DOS. If the sticky breakpoint did not exist, you could just enter **G**.

## MODEL IV TUTORIAL

This tutorial introduces you to the use of the Model IV real-time breakpoints and hardware trace buffer. Unlike the other tutorials, we won't be using a specific program, but will be setting breakpoints on system events.

### Step 1 — Configure Periscope for your system.

If you haven't done so already, run **SETUP** and **CONFIG** per the instructions in Chapter 4 to configure the Periscope software for your system.

### Step 2 — Install the Periscope software.

If you haven't done so already, make the Periscope directory the default directory, then enter **PS** from the DOS prompt.

### Step 3 — Display Periscope's screen.

Enter **RUN** with no arguments. This will display Periscope's screen.

### Step 4 — Set a hardware breakpoint.

We'll use the low word of the system timer for setting some hardware breakpoints. This memory location is updated by the system on each clock tick (every 55 milliseconds), so it is a convenient place to set breakpoints.

Enter **HM 0:46C L2 W** to set a hardware memory breakpoint on writes to the address 0:46C for a length of 2. Then, enter **GH** to go with hardware breakpoints enabled. You'll see the message **Setting breakpoints...** while Peri-

---

scope is programming the Model IV board and then the DOS screen will be briefly displayed. Then Periscope's screen is displayed, along with the message `EOI issued for IRQ 0`, since our breakpoint occurred during a hardware interrupt.

#### **Step 5 — View the hardware breakpoint event.**

Enter `HS` to see the event that caused the hardware breakpoint. You'll see one line of the trace buffer showing the address (`0046C`), the word value written (varies), the operation (`write`), the CPU cycle count in braces, the sequence number (`000`), and the string `Bottom`, indicating that this is the last entry in the trace buffer.

#### **Step 6 — Capture the breakpoint in the middle of the trace buffer.**

In many situations, it is helpful to have the breakpoint in the middle of the trace buffer, so that you can see what lead up to the breakpoint plus what happened afterward, all in real-time. Enter `HC TC;GH` to set the hardware controls to center the trace and go with hardware breakpoints enabled.

#### **Step 7 — View the trace buffer in Raw mode.**

You'll quickly return to Periscope's screen, but the `EOI issued for IRQ 0` message won't be displayed this time, since the hardware interrupt is no longer active due to the centering of the trigger. To get maximum space for the trace display, enter `/W` to turn off Periscope's windows. (You can restore the windows later using `Ctrl-F10`.) Now enter `HR` to look at the trace buffer in the raw format. This display shows the same sort of information as the `HS` display, which is the raw, undecoded trace history. Note the sequence numbers go from negative numbers to zero to positive numbers. The write to `0:46C` is at sequence number zero.

#### **Step 8 — View the trace buffer in Trace mode.**

Enter `T` to switch to trace mode. In this mode, Fetch cycles are disassembled and other CPU events such as memory and I/O reads and writes are shown interspersed in the

---

trace display. If you are debugging at the source level, your source code is displayed when the instruction matches the beginning of a source line. From the Periscope prompt, you can get directly to trace mode using the 'HT' command.

### **Step 9 – View the trace buffer in Unassembly mode.**

Enter U to switch to disassembly mode. In this mode, only Fetch cycles are disassembled and all other CPU events are not shown. From the Periscope prompt, you can get directly to this mode using the HU command. Now press the Esc key to return to the Periscope prompt.

### **Step 10 - Use selective capture.**

The Model IV trace buffer can be setup to capture just selected information. For example, to capture just the next 10H writes to 0:46C in the buffer, enter HC \* #10 S+;GH. This clears the hardware controls, sets the pass count to 10H, enables selective capture, and arms the board.

After approximately one second, Periscope's screen is displayed. Enter HR to display the buffer. You'll see 16 writes to 0:46C, with the value written increasing by one each time. Note that the HT or HU modes are not useful in this case, since there are no fetch cycles in the buffer.

### **Step 11 – Use a data breakpoint.**

The data (HD) and bit (HB) breakpoints may be used to qualify a memory or port breakpoint so that you can stop when a memory or port access occurs and the specified data value also occurs. For example, to set a breakpoint on the next time the letter P scrolls into the upper left-hand corner of the color display, enter the following commands:

```
HA *  
HM B800:0 L2 W  
HD LW 0750  
GH
```

This command sequence clears all hardware breakpoints (HA \*), sets a memory breakpoint on writes to B800:0, sets

---

a data breakpoint on the low word containing 0750, where 07 is the color attribute for gray on black and 50 is the character P, and then arms the board.

If you're using a monochrome display, substitute B000:0 for the address in the memory breakpoint. If you're using a 286 system, substitute W for LW in the data breakpoint.

**NOTE:** The command sequence above assumes that the screen's memory is accessed a word at a time. If you have problems with this example, enter `HD *;GH` to clear the data breakpoints and arm the board. Then after the hardware breakpoint occurs, look in the trace buffer to see how the memory was accessed and modify the `HD` command accordingly.

#### **Step 12 — Set a port breakpoint.**

To watch for the next write to port 20H, enter `HA *;HP 20 0;GH`. This command sequence clears all hardware breakpoints, sets a breakpoint on the next out to port 20H, and arms the board. When control is returned to Periscope, use the `HS` command to display the port write.

#### **Step 13 — Use sequential triggers.**

Periscope IV has a built-in state machine that lets you put together complex trigger conditions that are evaluated in real-time. For example, to stop on the first write to the timer word after the color display has been scrolled, use the following commands:

```
HA *  
HM B800:0 L2 W (0,1)  
HM 0:46C L2 W (1,1)  
GH
```

This command sequence first clears all hardware settings (`HA *`). It then sets a memory breakpoint on writes to the upper left-hand corner of the color display. When the breakpoint occurs the board moves from its starting state 0 to state 1 instead of immediately triggering. The third com-

---

mand sets a breakpoint on writes to 0:46C if and only if the board is currently in state 1. The exclamation point is used in the state settings to indicate a trigger condition.

For more information on Model IV's capabilities, please see Chapter 11 and the H-series commands in Chapter 15. ♦



# Configuring Periscope

## ■ Running SETUP and CONFIG

**T**his chapter describes the procedure for configuring Periscope for your computer system.

---

## RUNNING SETUP AND CONFIG

Before proceeding the first time, make a backup copy of the Periscope distribution disk, and store the original disk in a safe place. Then start at Step 1 below.

You no longer need to run CONFIG on a 'bare bones' system. You can now run it anytime after the initial setup by entering CONFIG from the Periscope directory, then following the steps below starting at Step 2.

### **Step 1- Place the Periscope backup disk in drive A, and enter A: SETUP.**

SETUP will prompt you for the name of the directory to be used by Periscope. The default is C:\PERI. SETUP then copies the Periscope files from the floppy disk to the indicated target directory. After the files have been copied, SETUP executes the CONFIG program, which configures Periscope for your system.

For best results, edit AUTOEXEC.BAT to contain the line SET PS=C:\PERI, or whatever directory you've chosen.

### **Step 2 - Answer the prompts on the full-screen display shown in Figure 4-1 (explanations follow ), then press F10.**

The configuration screen shows the current settings in PS1.COM. If you've never run CONFIG before (the answer to Periscope Model . . . is ?), the screen displays Periscope's default settings as shown in Figure 4-1. Otherwise it displays the previously-configured settings.

#### **Periscope Model (choose one): Enter choice (0-4)**

Enter the number corresponding to your model of Periscope.

**NOTE:** Periscope accesses port 70H to clear NMI. If you're debugging code that uses the CMOS data ports on an AT-class machine and have problems, configure Periscope as Model II-X,



---

```

Periscope Model (choose one):
 0 - Model I   (protected memory board & switch)
 1 - Model II  (break-out switch only, no board)
 2 - Model II-X (software only -- no NMI support)
 3 - Model III (hardware breakpoint board & switch)
 4 - Model IV  (hardware breakpoint board & CPU pod)
Enter choice (0-4) [1]
Minimum command length added to edit buffer (1-9) [1]
Characters between tab stops (1-8) [8]
Enable 386/486 debug registers (Y/N)? [Y]
Use fast color output (Y/N)? [Y]
System has a PC or XT motherboard with an 80286 or later turbo card (Y/N)? [N]
Use Watchdog timer on PS/2 to activate Periscope (Y/N)? [Y]
Ignore case of symbol names (Y/N)? [Y]
Point 286/386/486 exception interrupt 6 to Periscope (Y/N)? [Y]
Point 286/386/486 exception interrupt 0DH to Periscope (Y/N)? [Y]
Refresh interrupts 1, 2, and 3 each time Periscope is active (Y/N)? [Y]
Default to insert mode when Periscope's command-line editor is used (Y/N)? [N]
Use Periscope's menu system (Y/N)? [Y]
Function key use: Periscope [1], CodeView [2], or Turbo Debugger [3]? [1]

```

Key usage

Enter - next field  
 Home - first field  
 End - last field  
 Up - prior field  
 Down - next field  
 Esc - exit to DOS  
 F10 - configure PS

---

*Figure 4-1. Periscope Configuration Screen*

which does not use NMI.

### Minimum command length added to edit buffer (1-9)

The default value 1 adds all commands to the buffer. To add only commands longer than two characters, enter 3.

### Characters between tab stops (1-8)

Enter the tab column width, from 1 to 8 characters.

### Enable 386/486 debug registers (Y/N)?

Enter N if you're using a memory manager such as Compaq's CEMM that does not support real-mode access to the 80386 debug registers. Enter Y if you're using QEMM or 386MAX or no 386 control program.

### Use fast color output (Y/N)?

Enter N if you're using a 'snowy' color card.

### System has a PC or XT motherboard with an 80286 or later turbo card (Y/N)?

---

Enter **Y** if you're using a system with a PC or XT motherboard and an 80286, 80386, or 80486 turbo card.

**Use Watchdog timer on PS/2 to activate Periscope (Y/N)?**

Enter **Y** if you're using an IBM PS/2 and you want Periscope to automatically wake up after two seconds of 'missed' timer ticks. When the watchdog kicks in, Periscope displays the message **Grrr!**. To use this option, do not configure Periscope as Model II-X, since it does not support NMI and the watchdog requires NMI support.

**NOTE:** If you enter **Y**, do not use Ctrl-Alt-Del to reboot or your system will hang. Instead use Periscope's **QB**, **QS**, or **QL** commands to reboot your system, or install **PSKEY**, which disarms the watchdoggie.

**Ignore case of symbol names (Y/N)?**

Enter **N** if you want Periscope to support case sensitivity. If you enter **N**, Periscope will display **Periscope** in mixed case in the Watch window. Enter **Y** if you want Periscope to ignore the case of symbols. If you enter **Y**, Periscope will display **PERISCOPE** in all caps in the Watch window.

**Point 286/386/486 exception interrupt 6 to Periscope (Y/N)?**

Enter **N** if you're using software that uses this interrupt to intercept an illegal instruction. If possible, allow Periscope to handle this interrupt. It will help you capture runaway programs quickly.

**Point 286/386/486 exception interrupt 0DH to Periscope (Y/N)?**

Enter **N** if you're using software or hardware that uses this interrupt (IRQ 5). If you must use a mouse on this IRQ line, install the mouse before you install Periscope. If possible, allow Periscope to handle this interrupt.

**Refresh interrupts 1, 2, and 3 each time Periscope is active (Y/N)?**

---

Enter **N** if you don't want Periscope to ensure that these vectors point to it. This is useful if you want your program to filter INT 1, 2, or 3 before Periscope gets control.

**Default to insert mode when Periscope's command-line editor is used (Y/N)?**

Enter **Y** if you want Periscope to default to insert mode when commands are input.

**Use Periscope's menu system (Y/N)?**

Enter **N** if you want to turn off Periscope's menu system. When the menu system is enabled with **Y**, CONFIG also loads the on-line help and interrupt comments, consuming approximately 60KB. If you need to minimize Periscope's use of DOS memory, you may want to turn the menu system off.

**Function key use: Periscope (1), CodeView (2), or Turbo Debugger (3)?**

Periscope's function key usage is configurable. You can choose the standard Periscope function keys, the CodeView or Turbo Debugger emulation keys, or configure your own keys. For technical details, see the file NOTES.TXT. ♦





# Installing Model I Hardware

- **Checking for Conflicts with the Model I, Rev 3 Board**
- **Setting the DIP Switches on the Model I, Rev 3 Board**
- **Populating the Model I, Rev 3 Board**
- **Installing the (Model I, Rev 3) Plus Board with Model IV**
- **Installing the Model I, Rev 3 Board**
- **Populating the Model I/MC Board**
- **Installing the Model I/MC Board**

**I**f you have the Periscope I, Rev 3 board (for AT-compatible machines), you'll find the information you need to install your board and break-out switch in the first five sections of this chapter. If you have the Periscope I/MC board (for machines with the Micro Channel bus architecture), you'll find the information you need to install your board and break-out switch in the last two sections of this chapter.



---

## **CHECKING FOR CONFLICTS WITH THE MODEL I, REV 3 BOARD**

The Model I, Rev 3 board uses 32K of memory space and two consecutive I/O ports. When you receive your board, the DIP (Dual In-line Package) switches are set to use memory starting at D000:0000 and I/O ports starting at 300H. For proper operation, the memory and ports used by the board must not be used by any other device.

The area of memory used by the board is above DOS memory and the area reserved for screen buffers. The board's default setting may conflict with other add-on memory cards that use this area for a RAM disk, ROM device driver, or EMS board. Check the documentation for any non-standard expansion cards to see if this is the case. If you're not sure of the memory usage in your system, enter `PSTEST /A` at the DOS prompt to display the high-memory usage.

The I/O ports used by the board is 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. These ports conflict with some network cards and some tape backup cards, which use ports 300H through 30FH. If you have one of these cards, try using port 310H. Other expansion options may also use Periscope's default I/O ports. Check the documentation for any non-standard expansion cards 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 your Periscope board, skip the next section on setting the DIP switches.

## **SETTING THE DIP SWITCHES ON THE MODEL I, REV 3 BOARD**

There are two DIP switches on the Model I, Rev 3 board. The eight-position switch labeled SW1 controls the I/O

ports used by the board. The four-position switch labeled SW2 controls the starting address of memory used by the board.

**Setting I/O Port Addresses.** SW1 is preset to use I/O ports starting at 300H. Switch SW1 may be set to indicate any I/O ports on a four-byte boundary. Set the switches so that they do not conflict with other ports in the system. Certain ports are off-limits, such as zero to 100H and ports already in use by another expansion card. If port 300H is not available, try 310H. Consult the IBM Technical Reference Manual and documentation for your non-standard expansion cards to avoid conflicts. Also see the list of port usage in the file NOTES.TXT.

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

\* N/A WITH MODEL IV

Figure 5-1. Setting Dip Switch SW1 on All Boards

The I/O port address can be read using the table shown in

Figure 5-1. The first section of the table illustrates the use of switch positions S1 and S2 to set the 'units' part of the address; the second section of the table illustrates the use of switch positions S3, S4, S5, and S6 to set the 'tens' part of the address; and the third section of the table illustrates the use of switch positions S7 and S8 to set the 'hundreds' part of the address.

**Setting Memory Addresses.** SW2 is preset to use memory starting at D000:0000. You must not set the switch to conflict with other memory installed in the system. Certain ranges of memory that can be used by the board are off-limits on some systems. For instance, A000:0000 to A000:FFFF is used by EGAs and VGAs; B000:0000 to B000:7FFF is used by monochrome and monochrome graphics adapters; B800:0000 to B800:7FFF is used by color displays (CGA, EGA, and VGA); C000:0000 to C000:3FFF is used by EGAs; C000:0000 to C000:7FFF is used by some VGAs; D000:0000 to D000:FFFF is used by EMS boards and by some network cards; E000:0000 to E000:FFFF is available on PCs and XTs and most 386 systems but not on ATs.

Starting Address	S1	S2	S3	S4	Comments
A000:0 (640 K)	N/A	ON	ON	ON	No EGA or VGA
B000:0 (704 K)	N/A	OFF	ON	ON	No mono display
B800:0 (736 K)	N/A	OFF	OFF	OFF	No color display
C000:0 (768 K)	N/A	ON	OFF	ON	No EGA or VGA
C400:0 (784 K)	N/A	ON	ON	OFF	No VGA, no XT
C800:0 (800 K)	N/A	ON	OFF	OFF	EGA/VGA & EMS OK, no XT
D000:0 (832 K)	N/A	OFF	OFF	ON	No EMS board
E000:0 (896 K)	N/A	OFF	ON	OFF	PC OK, no AT

*Figure 5-2. Setting Dip Switch SW2 on Model I, Rev 3*

If memory starting at segment D000:0000 is already in use, try using E000:0000 if the system is a PC or an XT. If the system is an AT or 386 system, try using C800:0000. On an XT-class machine, do not use C400:0000 or C800:0000, since these settings will conflict with the hard disk controller BIOS, which is located at C800:0000. If you use 386MAX, or another memory manager, be sure to use the RAM= statement or equivalent to keep the memory manager from using memory where Periscope's memory is located.



---

Figure 5-2 shows the eight possible switch settings. Note that switch position 1 is ignored; only positions 2 through 4 are used.

**NOTE:** This manual describes the Model I, Rev 3 board, released in September, 1988. For complete information on earlier Model I boards, see the versions of the manual that accompanied those boards. See the NOTES.TXT file for a description of all Model I boards.

## POPULATING THE MODEL I, REV 3 BOARD

If you have a Model I, Rev 3 board with zeroK of memory, you can populate the board to 512K or to one megabyte with 16 or 32 DRAM chips respectively. If you have a 512K board, you can populate it to one megabyte by adding 16 DRAM chips. The chips should be 256K by 1 dynamic RAMs with an access time of 150 NS or faster. The generic part number is 41256-15, although different manufacturers may use different part numbers.

To install the chips, place the board on a flat anti-static surface, with the mounting bracket to your right. If you do not have an anti-static work surface, use the conductive bag in which the board was shipped. For one megabyte, populate all four columns (U1-U32). For 512K, populate the two columns nearest to the mounting bracket (U17-U32). See Figure 5-3.

**NOTE:** The board must have exactly two or four columns populated. It cannot be run with either one or three columns of RAM!

Be sure that the notch on the chips points away from the mounting bracket on the board. After inserting each chip, ensure that the IC is securely seated, with no bent pins.

Now continue with the installation procedures. Be sure to run PSTEST to confirm proper operation of the board.



---

# INSTALLING THE (MODEL I, REV 3) PLUS BOARD WITH MODEL IV

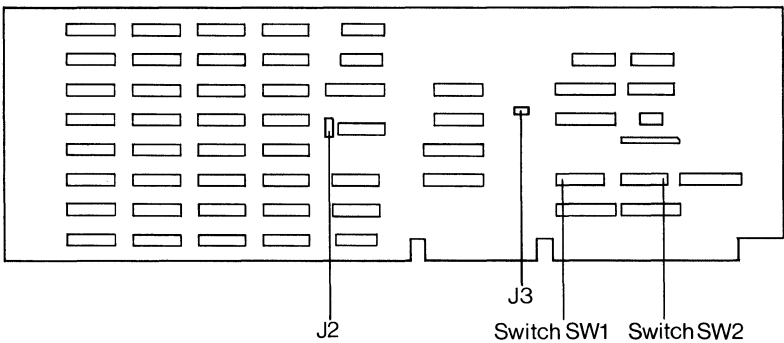
The Model I, Rev 3 board can be used with a Periscope IV board to provide write-protected memory so that all of Periscope's code, data, and tables are running out of the protected memory instead of the lower 640K of system memory. The Model I board is called a "Plus" board when used with Model IV.

To use this combination, install both boards in your system, setting the Periscope IV board to use the same starting port as the Periscope I board. For example, if you've set the Periscope I, Rev 3 board to start at port 310H, set the Periscope IV board to start at port 310H, too. Configure Periscope as Model IV.

## INSTALLING THE MODEL I, REV 3 BOARD

The board has a 16-bit connector for use in an AT-class machine. It can also be used in a PC- or XT-class machine. If the board detects that it is in an 8-bit slot, it runs in 8-bit mode. If it is in a 16-bit slot, it runs in 16-bit mode, unless jumper J3 is removed.

To run in as many systems as possible, the board has a wait-



*Figure 5-3. Layout of the Model I, Rev 3 Board*

---

state generator that controls how many wait states are inserted for memory accesses. Jumper J2 is used to control the number of wait states. See both Figure 5-3 and the later section on jumpers for details on setting these jumpers.

The Model I board works fine on most systems regardless of the CPU speed. Since it is a bus-based device, it is affected by the bus speed, which is usually 8MHz.

**NOTE:** If you're using a system that has a Chips and Technologies chip set, be sure to disable Video ROM shadowing for proper operation of Periscope I.

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 up your system.**

Remove the cover mounting screws on the rear of the system unit. Slide the cover of the system unit forward and remove the cover from the system unit. **Be sure to ground yourself. While working inside the system unit, touch the chassis or other ground frequently to avoid the build up of static electricity.**

### **Step 2 - Install the Model I, Rev 3 board in an empty slot.**

The board can be installed in any full-length slot. If you're using a PC- or XT-class machine, be sure to leave the plastic cover on the 16-bit bus connector so that the gold fingers won't accidentally contact anything on the motherboard. In an AT or 386, be sure to remove the plastic cover and install the board in a 16-bit expansion slot.

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.

Align the board with the expansion slot and lower it until the edge connector is resting on the expansion slot recep-

---

tacle on the system board. Press the board straight down until it seats in the expansion slot. Check the fit of the mounting bracket. If it is not correctly aligned with the back panel, adjust it by loosening the two screws that attach the bracket to the board.

Install the retaining screw through the board's bracket into the PC's back panel and tighten it. Be sure the board is fully inserted in the slot. On some systems, the bottom edge of the Periscope I board may run into some tall chips mounted on the motherboard. Make sure that there's no chance of a short-circuit between the Periscope I board and any other components. If you have a problem with this, we have socket extenders available.

**Step 3 - Close your system up.**

Replace the cover of the system unit and install the cover mounting screws.

**Step 4 - Install the remote break-out switch.**

If you plan to use the remote break-out switch, plug it into the phono jack on the board's mounting bracket now, while the power is still off.

**Step 5 - Re-connect all peripherals and replace the power cord.**

**Step 6 - Run the Periscope I board's diagnostics.**

Boot the system and execute the program PSTEST to confirm the proper operation of the board. Be sure to add the /M or /P options needed if the memory or ports have been changed from the default values. If you get any errors, try adding some wait states using jumper J2. If you still get errors, try forcing the board to 8-bit mode by removing jumper J3. For more information, see the description of PSTEST in Chapter 10.

**Step 7 - Install the Periscope software.**

Enter PS to install Periscope, then press the break-out

---

switch. The Periscope screen should be displayed. To continue, enter G and press return. Do not press the break-out switch before the Periscope software 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 on a PC or XT, 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.

### **Jumpers on the Model I, Rev 3 Board**

There are two jumpers of interest on the Model I, Rev 3 board, shown in Figure 5-3 and described below.

**J2 - WAIT-STATE GENERATOR.** When the jumper J2 is not connected across two of the three pins, zero wait states are added. When the jumper is across the top two pins (the two furthest away from the gold fingers), one wait state is added. When the jumper is across the bottom two pins, two wait states are added. You should run the board with as few wait states as possible for best performance. The wait-state setting on the Model I board does not affect anything other than Periscope's memory.

**J3 - 8-BIT OPERATION.** Removal of jumper J3 forces 8-bit operation, even in a 16-bit slot. You should avoid 8-bit operation when possible, since Periscope runs much slower than when using 16-bit mode.

## **POPULATING THE MODEL I/MC BOARD**

If you have a Periscope Model I/MC board with 512K, you can populate the board to 1 megabyte, 1.5 megabytes, or 2 megabytes with 4, 8, or 12 DRAM chips respectively. The chips needed are 256K by 4 dynamic RAMs with an access time of 120 NS or faster. The generic part number is 44256-12, although different manufacturers may use different part numbers.

To install the chips, place the board on a flat anti-static surface, with the mounting bracket to your right. If you do not have an anti-static work surface, use the conductive bag in



---

which the board was shipped. For one megabyte, populate the two columns nearest the mounting bracket. For 1.5 megabytes, populate the three columns nearest to the mounting bracket. For two megabytes, populate all four columns.

Be sure that the notch on the chips points away from the mounting bracket on the board. After inserting each chip, ensure that the IC is securely seated, with no bent pins.

Now continue with the installation procedures. Be sure to run PSTEST to confirm proper operation of the board.

## **INSTALLING THE MODEL I/MC BOARD**

**Step 1 - Copy the Periscope Adapter Description File (@6077.ADF) to the backup copy of your reference disk.**

Insert a backup copy of your reference disk in Drive A and boot the system. When the menu is displayed, select **Copy an option Diskette**. When prompted, remove the backup copy of the reference disk and insert the Periscope disk in Drive A. When prompted, remove the Periscope disk and re-insert the backup copy of your reference disk.

**Step 2 - Open up your system.**

Before installing the board, be sure that the power is off and that the power cord is removed. Open the PS/2 by removing the cover from the system unit. **Be sure to ground yourself. While working inside the system unit, touch the chassis or other ground frequently to avoid the build up of static electricity.**

**Step 3 - Install the board in an empty slot.**

The board can be installed in any slot. Select an available expansion slot and loosen the screw fastening the dummy metal bracket, using a small screwdriver if necessary. Remove the dummy bracket, but save it in case you ever remove the Periscope board.

---

Align the board with the expansion slot card guides and lower the board until you encounter resistance. Make sure the notch in the metal bracket on the back edge of the board straddles the bracket retaining screw, then press the board straight down until it seats in the expansion slot. Be sure the board is fully inserted in the slot. Tighten the retaining screw you previously loosened.

**Step 4 - Install the remote break-out switch.**

If you plan to use the remote break-out switch, plug it into the phono jack on the board's mounting bracket now, while the power is still off.

**Step 5 - Replace the cover of the system unit.**

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

**Step 7 - Boot the system with the backup copy of your reference disk in Drive A.**

Error 165 will be displayed. This error is normal. It means that the Periscope I/MC board is installed, but the system doesn't yet know about it.

**Step 8 - Configure the system to include the Periscope board.**

Run the system configuration program. When done, remove the copy of your reference disk from Drive A and reboot the system from your hard disk.

**Step 9 - Run the Periscope I board's diagnostics.**

Execute the program PSTEST to confirm the proper operation of the board. The memory and port addresses used by the Periscope board will be displayed by PSTEST. For more information, see the description of PSTEST in Chapter 10.

**Step 10 - Run Periscope.**



---

Execute Periscope (enter **PS**), then press the break-out switch. The Periscope screen should be displayed. To continue, enter **G** and press return. Do not press the break-out switch before the Periscope software is installed. You'll get a parity error and have to turn the power off and back on. ♦



# Installing Model II Hardware

## ■ Installing the Model II Break-Out Switch

**I**f you have Periscope Model II, you'll find the information you need to install your remote break-out switch in this chapter.



---

## INSTALLING THE MODEL II BREAK-OUT SWITCH

Before you install 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 Figure 6-1. 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 your system.

Remove 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. **Be sure to ground yourself. While working inside the system unit, touch the chassis or other ground frequently to avoid the build up of static electricity.**

### Step 2 - Route the connector end of the cable assembly into your system from the back of the unit.

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 in Figure 6-1) and the other is a gold-plated probe (see Exploded View B in Figure 6-1).

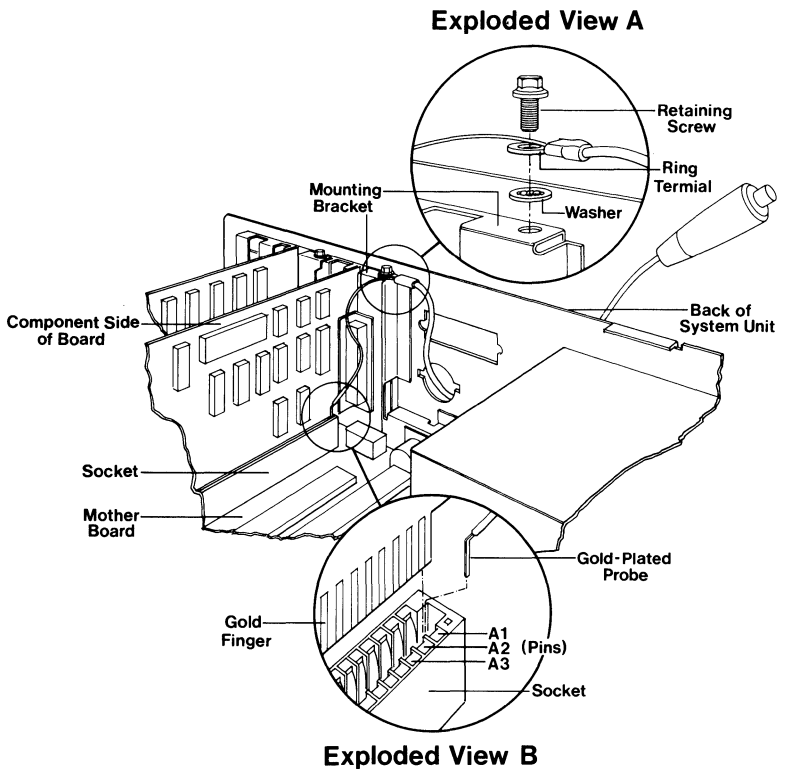
There are several possible ways of routing the cable into the PC, either through a knock-out panel or in the space between the keyboard connector and the expansion slots. Do NOT install the cable so that it is lying on top of the back panel. When the cover is installed, the cable may be crimped and possibly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

### Step 3 - Select an in-use slot, then install the ring terminal for ground and strain relief.

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. If the slot nearest to the power supply 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 Figure 6-1. 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.



*Figure 6-1. Installation of Model II Break-out Switch*



---

#### **Step 4 - Install the gold-plated probe.**

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 the probe so that it 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 Figure 6-1. Push the probe in as far as possible to ensure a good connection and to keep the non-insulated part of the probe from contacting anything other than the desired pin.

**NOTE:** Not all boards have a gold finger at pin A1. Look for the first socket connector to positively identify the pin.

If you can't get the probe into the socket, try removing the board and sliding the board and the probe in together. On systems without enough room to get your hands near the socket, you may need to use a pair of needle-nose pliers.

#### **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 your machine and install the cover mounting screws.**

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

**Step 9 - Install the Periscope software.**

Enter **PS** to install Periscope, then press the break-out switch. The Periscope screen should be displayed. To continue, enter **G** and press return. Do not press the break-out switch before the Periscope software 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 on a PC or XT, 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 Model IV Hardware

- Checking for Conflicts
- Setting the DIP Switch
- Model IV Components
- Installing the Model IV Hardware

**I**f you have Periscope Model IV, you'll find the information you need to install the hardware in this chapter.

---

## CHECKING FOR CONFLICTS

The Model IV board uses no memory and eight consecutive I/O ports. For proper operation, the ports used by the board must not be used by any other device.

When you receive your board, the DIP (Dual In-line Package) switch is set to use I/O ports starting at 300H. These ports 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.

These ports conflict with some network cards and some tape backup cards, which use ports 300H through 30FH. If you have one of these cards, try using port 310H. Other expansion options may also use Periscope's default I/O ports. Check the documentation for any non-standard expansion cards 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 supports the ten low-order bits of a port address.

If you find no conflicts with the I/O ports used by your Periscope board, skip the next section on setting the DIP switch.

## SETTING THE DIP SWITCH

The eight-position DIP switch on the Model IV board, labeled SW1, controls the I/O ports used by the board. SW1 is preset to use I/O ports starting at 300H. It may be set to indicate any I/O ports on an eight-byte boundary. Set the switch so that it does not conflict with other ports in the system. Certain ports are off-limits, such as zero to 100H and ports already in use by another expansion card. If port 300H is not available, try 310H. Consult the IBM Technical Reference Manual and documentation for your non-standard expansion cards to avoid conflicts. Also see the list of port usage in the file NOTES.TXT.

The I/O port address can be read using the table in Figure 5-1. The first section of the table illustrates the use of switch positions S1 and S2 to set the 'units' part of the address; the second section of the table illustrates the use of switch positions S3, S4, S5, and S6 to set the 'tens' part of the address;



---

and the third section of the table illustrates the use of switch positions S7 and S8 to set the 'hundreds' part of the address.

Since the Model IV board uses eight consecutive I/O ports, the I/O port address indicated by SW1 on the Model IV board must be evenly divisible by eight. Referring to the table in Figure 5-1, ignore switch position S1, since it has no effect on the Model IV board.

**NOTE:** If you're using the Plus (Model I) board with your Model IV, use the same starting port on both boards. See Chapter 5 for installation instructions.

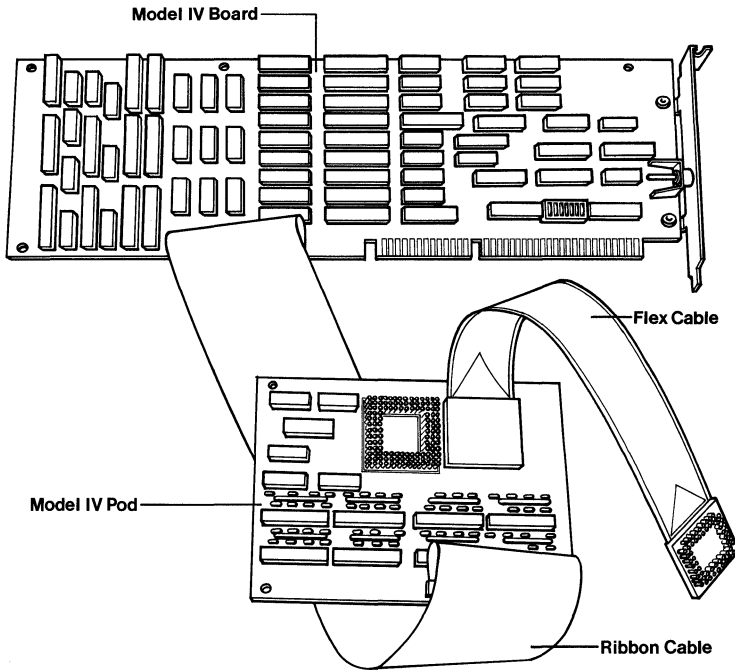
## MODEL IV COMPONENTS

Periscope IV has these four hardware components:

- The full-length board labeled 'Periscope Model IV' (referred to as the 'Model IV Board');
- The small board labeled 'Periscope Model IV 286/386 Pod' (referred to as the 286/386 'Pod');
- A 50-conductor ribbon cable with 2x25 connectors on each end (referred to as the 'Ribbon Cable');
- A flexible printed circuit cable for either the 80286 or 80386 CPU with Pin Grid Array (PGA) connectors on each end (referred to as the 'Flex Cable').

**NOTE:** There is an alternate 386 Pod (386 Pod Rev 1) for use in machines in which the 286/386 Pod will not work. This 386 Pod is smaller than the 286/386 Pod and does not require the Flex Cable described above. It plugs directly into the CPU socket. If you have the 386 Pod, follow Steps 1 through 6 of the installation instructions below. Then install the 386 Pod per Steps 6a through 6c. Skip Steps 7 through 11, and resume at Step 12 of the instructions. For more information about the pods, see the file NOTES.TXT.

The Model IV Board is connected to the Pod via the Rib-



*Figure 7-1. Periscope Model IV*

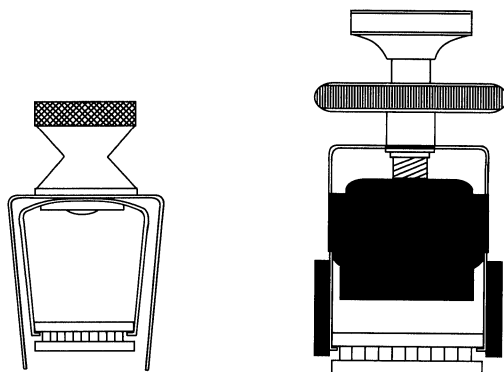
bon Cable. The 286/386 Pod is connected to the CPU via the Flex Cable. Figure 7-1 shows an assembled Model IV with a standard 286/386 Pod and a 286 Flex Cable.

## INSTALLING THE MODEL IV HARDWARE

Before installing the hardware, 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. You'll also need both the PGA chip puller shipped with the Flex Cable or Pod to remove your CPU (see Figure 7-2) and the small 9cm (3.5-inch) metal "crowbar" (see Figure 7-3).

The crowbar tool has a number of uses:

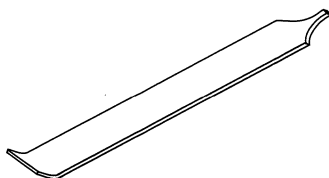
- Use either end to help remove the Ribbon Cable



*Figure 7-2. 286 (left) & 386 (right) Chip Pullers*

without bending any pins;

- Use its wide end to remove the Flex Cable from the Pod or to remove the CPU chip from its socket;
- Use its narrow end to remove the CPU chip from the Flex Cable. Do not use it at the corners of the CPU, because you can break off pieces of the ceramic material.



*Figure 7-3. Crowbar*

**NOTE:** Follow the steps below in order. The installation of Model IV is fairly complex; if you skip a step or skim the instructions, you may damage the Model IV hardware or your computer system!

---

If possible, have a co-worker read the instructions to you while you perform the installation. This technique helps you avoid losing your place and reduces the chances of making an expensive mistake.

If you don't want to do the installation, we'll be happy to install and test your unit for you. Please call our sales number for details.

### **Step 1 - Open the PC.**

Remove the cover mounting screws on the rear of the system unit. Slide the cover of the system unit forward and remove it.

### **Step 2 - Ground yourself.**

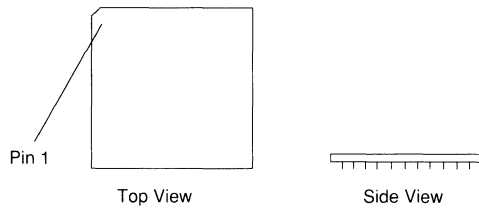
While working inside the system unit, be sure to frequently ground yourself (touch the chassis or other ground) to avoid the build up of static electricity. Any time you move your feet, be sure to ground yourself again. If you have an 80386 system, be especially careful, since these chips are expensive and can be hard to find.

### **Step 3 - Locate the CPU chip.**

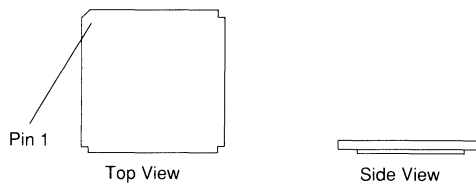
Make sure the CPU is socketed. If it is soldered directly to the motherboard, you cannot use Periscope IV.

If the system has an 80386, the CPU is always a PGA (Pin Grid Array) package. If the system has an 80286, the CPU may be one of three packages: PGA, PLCC (Plastic Leaderless Chip Carrier), or LCC (Leaderless Chip Carrier).

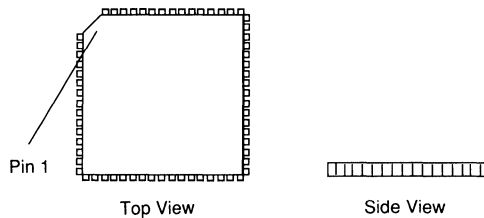
The **PGA chip** (see top drawing in Figure 7-4) is made of ceramic material with no metal visible around its edges. When viewed from the top, the socket is entirely hidden by the chip. Notice the notch at the Pin 1 location, and the shape of the other three corners of the chip. When viewed from the side, the PGA chip has a profile, i.e., it sits above its socket when installed.



Pin Grid Array (PGA)



Leaderless Chip Carrier (LCC)



Plastic Leaderless Chip Carrier (PLCC)

*Figure 7-4. 80286 CPU Packages*

The **LCC chip** (see middle drawing in Figure 7-4) is made of ceramic material with no metal on the edges of the chip itself. When viewed from the top, the socket is visible around the chip. Notice the notch at the Pin 1 location, and the shape of the other three corners of the chip. When viewed from the side, the LCC chip lies flat in its socket, i.e., it has no profile. The socket usually has a retainer clip to hold the CPU in the socket.

The **PLCC chip** (see bottom drawing in Figure 7-4) is made

---

of plastic with metal pins visible around the edges of the chip. When viewed from the top, the socket is visible around the chip. Notice the notch at the Pin 1 location, and the shape of the other three corners of the chip. When viewed from the side, the PLCC chip lies flat in its socket, i.e., it has no profile.

The Flex Cable requires a PGA connection. If you have a PLCC or an LCC CPU, you'll need an adapter to convert your CPU connection to a PGA. If you have an adapter, you will need to pull your CPU and install the adapter (see manufacturer's instructions) before you can install the Model IV Flex Cable. If you do not have an adapter, please call The Periscope Company. Both PLCC and LCC adapters are available.

#### **Step 4 - Get access to the CPU.**

Remove expansion boards as needed to allow access to the CPU chip. In some systems, the CPU chip is underneath the disk drive cage. If this is the case, you'll need to remove the motherboard to be able to complete the installation.

**NOTE:** Once the Flex Cable is installed, it will add 6.4mm (one quarter inch) to the total height of the CPU. You may have to change the placement of expansion boards to make clearance for the higher CPU.

#### **Step 5 - Find pin 1 of the CPU.**

During the installation of the Flex Cable, you'll need to know which corner of the CPU is pin 1. Some motherboards are marked, but for those that aren't, find the notched (or dotted) corner of the CPU chip and mark the motherboard with a felt-tip pen or use one of the adhesive 'dots' shipped in the Model IV board package.

#### **Step 6 - Use the PGA chip puller to remove the CPU.**

Figure 7-2 shows the puller included with the 80286 Flex Cable kit for the 80286 CPU chip (left) and the puller included with the 80386 Flex Cable kit for the 80386 CPU

---

chip (right). A chip puller is also included in the 386 Pod Kit.

**NOTE:** Be sure to insert the chip puller between the CPU and its socket, not between the CPU's socket and the motherboard!

**If you have an 80286,** unscrew the chip puller's knob to take the chip puller apart. Carefully slide the curved edges of the piece with the screw threads between the CPU and its socket. Center it in the middle of two opposite sides of the CPU, not the corners. Figure 7-2 shows the 80286 chip puller holding the CPU in position for pulling.

Now, slide the C-shaped piece over the piece holding the CPU chip. Tighten the knob. When you feel resistance, turn the knob very slowly so the CPU is extracted gradually. The first time a CPU is removed from its socket can require an amazing amount of force. If the CPU is not cooperating, rotate the chip puller 90 degrees and try again, working a little bit further each time. Also, if there are obstructions on the motherboard, turn the chip puller 90 degrees to avoid the obstructions.

Ground yourself, then remove the CPU from its socket. Check the CPU for any bent pins. Use needle-nose pliers to carefully straighten any bent pins you find. Now place the CPU on a flat anti-static surface.

**If you have an 80386,** see the Augat Operating Instruction sheet included with the chip puller. The following directions may clarify the manufacturer's instructions.

Turn the chip puller's black knob in a counter-clockwise direction until the "teeth" are lowered to the bottom of the legs. Spread the teeth by gripping underneath the black knob with your forefinger and middle finger and pushing down on the smaller knob with your thumb. Slide the teeth between the CPU and its socket, centering it in the middle of two opposite sides of the CPU. Figure 7-2 shows the 80386 chip puller holding the CPU in position for pulling.

Make sure that the legs of the tool do not interfere with any

---

components on the motherboard. You may need to turn the chip puller 90 degrees to avoid other components. Also, make double sure that the teeth are between the CPU socket and the CPU as shown in Figure 7-2, not between the motherboard and the CPU socket.

**NOTE:** Should this chip puller not work with your 80386 chip, we have another puller available. Please call Tech Support for more information.

Now, turn the black knob in a clockwise direction. When you feel resistance, turn the knob very slowly so the CPU is extracted gradually. The first time a CPU is removed from its socket can require an amazing amount of force. If the CPU is not cooperating, rotate the chip puller 90 degrees and try again, working a little bit further each time.

Ground yourself, then remove the CPU from its socket. Check the CPU for any bent pins. Use needle-nose pliers to carefully straighten any bent pins you find. Now place the CPU on a flat anti-static surface.

#### **Step 6a - Insert the CPU into the 386 Pod.**

**NOTE:** If you have the 286/386 Pod and Flex Cable, go directly to Step 7 now. If you have the 386 Pod (without the Flex Cable), perform Steps 6a, 6b, and 6c, then go to Step 12 of the instructions.

Ground yourself, then insert the CPU into the Pod after confirming that the dot and notch on the CPU are aligned with the notch on the Pod's silkscreen (this aligns pin 1 on Pod and CPU). Before pushing the CPU fully into position, look on the underside of the chip for any bent or misaligned pins. Then firmly and evenly press the CPU into the socket. The exposed CPU pin length between the bottom of the CPU and the top of the socket should be about one millimeter (1/20 inch).

#### **Step 6b - Insert the 386 Pod into the CPU socket on the motherboard.**

The Pod is shipped with one spacer PGA socket already in-



---

stalled on the bottom of the Pod and one spare socket in the package. In most cases, you won't need the spare socket.

Remove the protective foam from the bottom of the Pod. Check the pins on the socket. If any are bent, carefully straighten them with needle-nose pliers. Align pin 1 of the CPU in the Pod with pin 1 of the motherboard socket. Using a bright light source such as a flashlight, insert the Pod into the motherboard. Check for any bent or misaligned pins and then firmly press the Pod into the socket. If the Pod runs into any obstructions on the motherboard, remove it and add the spare spacer socket.

**Step 6c - If you removed the motherboard to install the Pod, reinstall it in the system unit now.**

**NOTE:** Be very careful to correctly align pin 1 on the CPU, pin 1 on the Pod, and pin 1 on the motherboard. Failure to do so can damage the CPU chip.

Now continue at Step 12 below.

**Step 7 - Insert the CPU into the Flex Cable.**

The Flex Cable has different connectors on each end. The end with the black cap will be plugged into the Pod. Hold the cable so that the end with the black cap is on your left and the other end is on your right. The CPU is plugged into the right end of the cable. Note the notch on the top right corner of the socket. This is pin 1. It must be aligned with pin 1 on the CPU chip.

**NOTE:** Pin 1 is on the bottom right corner of the socket on the 80286 Flex Cable.

The Flex Cable is not an ordinary cable. It is an expensive, high-technology controlled impedance flexible printed circuit. It cannot take the rough treatment that a ribbon cable or other cable can. The Flex Cable is very reliable if it is not abused. To get the best service from the cable:

- Install the Flex Cable in a system and leave it there. If

---

you need to move Periscope IV from one system to another, install a Flex Cable in each system and move the Model IV Board and Pod from system to system.

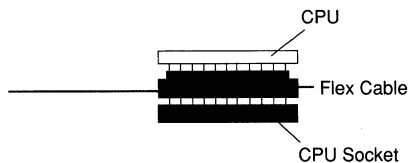
- Don't bend or twist the cable any more than necessary.
- Don't crease the cable unless it is absolutely required. If you must crease the cable, NEVER uncrease it. Uncreasing the cable will break the circuit traces.
- Do not put anything on the cable.
- Do not bend the cable's pins. If they are bent, carefully straighten them with needle-nose pliers.

Remove the rubber band holding the protective foam on the CPU end of the Flex Cable, and hold the foam in place. Carefully align the CPU's pins with the socket on the Flex Cable. Before pressing the CPU into the socket, examine it from all four sides to be sure that all pins are correctly aligned.

If you find any bent pins, straighten them with needle-nose pliers. Gently but firmly press the CPU and the socket together. Be sure to hold the protective foam on the Flex Cable's pins during this process so that you don't bend the pins on the Flex Cable. For proper operation, the exposed CPU pin length between the bottom of the CPU and the top of the socket should be about one millimeter (1/20").

**NOTE:** Be very careful to correctly align pin 1 on the CPU, pin 1 on the Flex Cable, and pin 1 on the motherboard. Failure to do so can damage the CPU chip.

**Step 8 - Install the CPU/Flex Cable assembly back into the motherboard.**



*Figure 7-5. Cross-section: CPU/Flex Cable/Motherboard*

---

First, remove the protective foam from the Flex Cable's pins on the CPU end. Check the pins on the Flex Cable socket. If any are bent, carefully straighten them with needle-nose pliers. Align pin 1 of the Flex Cable socket and pin 1 of the motherboard socket. Gently but firmly press the Flex Cable assembly into the socket on the motherboard.

If you need to get more clearance between the Flex Cable and the motherboard, please call Tech Support. We can supply you with a PGA socket that can be used as a spacer socket between the motherboard and the socket on the Flex Cable.

Now triple check that pin 1 is correctly aligned on all three pieces, the CPU, the Flex Cable socket, and the motherboard. If this is not done correctly, you can damage an expensive CPU chip.

**Step 9 - If you removed the motherboard to install the Flex Cable, reinstall it in the system unit now.**

**Step 10 - Check the setting of jumper J1 on the Pod.**

Jumper J1 is on the edge of the pod near the 80286 socket (see Figure 7-6). If you're using an 80386 system, the jumper should be connected across the two pins (default). If you're using an 80286 system, the jumper should not be connected across the two pins.

**Step 11 - Install the Flex Cable in the Pod, then choose a location for the Pod.**

Remove the protective foam from the Pod end of the Flex Cable and insert the Flex Cable into the appropriate socket on the Pod. Be sure to check for bent pins and align pin 1 on both connectors. When correctly aligned, the Flex Cable is at a right angle to the Pod and pointing away from the Pod (see Figure 7-5).

**NOTE:** There should be an audible click when you insert the Flex Cable into the Pod.

---

Use the black plastic box for insulating the Pod. You may leave the cover on so that the Pod is fully enclosed, or take the cover off so that the box becomes a tray for the Pod.

When you choose a location for the Pod, keep the following points in mind:

- It must be able to connect to the Flex Cable without undue bending or creasing of the cable.
- The Pod must be securely positioned so that it won't move around after installation. Use the Velcro included with the Pod to secure it as needed.
- It should be positioned so that you can put the system unit cover back on.

In some systems there's room on top of the disk drive cage for the Pod. In other systems you can lay the Pod across the tops of expansion boards or between the boards and still have enough clearance to reinstall the cover. In some systems, you'll just have to leave the cover of the system unit off.

#### **Step 12 - Plug the Ribbon Cable into the Model IV Board at connector J1.**

See Figure 7-8. Note that the cable is keyed, so that it can be installed in only one direction. When removing the Ribbon Cable, use the crowbar and be careful not to bend the pins.

#### **Step 13 - Install the Model IV Board.**

The board can be installed in any one of the available full-length slots on the system board, either in an 8-bit or in a 16-bit slot. If you're installing the board in a 16-bit slot, remove the plastic cover from the high bus connectors. If you're installing the board in an 8-bit slot, leave the plastic cover on the high bus connectors so that the exposed fingers won't cause a short-circuit.

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.

Align the Periscope IV Board with the expansion slot and lower it until the edge connector is resting on the expansion slot receptacle in the system board. Press the board straight down until it seats in the expansion slot. Check the fit of the mounting bracket. If it is not correctly aligned with the back panel, adjust it by loosening the two screws that attach the bracket to the board. Install the retaining screw through the board's bracket into the PC's back panel and tighten it.

Be sure the board is fully inserted in the slot. On some systems, the bottom edge of the Model IV Board may run into some tall chips mounted on the motherboard. Make sure there's no chance of a short-circuit between the Model IV Board and any other components. If you have a problem, we have socket extenders available. Call tech support for details.

#### **Step 14 - Plug the Ribbon Cable into the Pod at connector J7.**

This does not apply if you have the 386 Pod since its ribbon cable is permanently attached. See Figure 7-6.

#### **Step 15 - Re-connect all peripherals and replace the power cord.**

#### **Step 16 - Turn on the power to the system.**

Watch the fan in the power supply when you turn the power on. If the fan does not run normally, turn the power off immediately. Check the alignment of the CPU and Flex Cable, since a short circuit of some sort has probably occurred.

#### **Step 17 - Replace the cover of the system unit and install the cover mounting screws.**

#### **Step 18 - Install the remote break-out switch.**

If you plan to use the break-out switch, plug it into the phono jack on the Model IV board's mounting bracket now, while the power is still off.

---

**NOTE:** When using both the Model IV Board and a Model I board, connect the break-out switch to the Model IV Board. Otherwise, you'll get the Parity error 2 message when the break-out switch is pressed.

### **Step 19 - Run the Model IV diagnostics.**

Boot the system and execute the program PS4TEST to confirm the proper operation of the board (see Chapter 10). Be sure to add the /P option needed if the ports have been changed from the default values.

If the system does not boot or behaves erratically, re-check all connections. On 286 systems, you may need to change resistor R1 on the Pod using the resistor kit that is included in the 286 Flex Cable Kit.

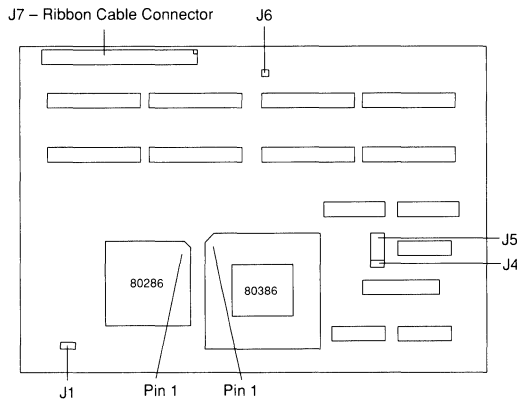
### **Step 20 - Install the Periscope software.**

Enter PS to install Periscope, then press the break-out switch. Periscope's screen should be displayed. To continue, enter G and press return.

**NOTE:** If you need to remove the 286/386 Pod but don't want to remove the Flex Cable from the CPU, lay the Pod end of the Flex Cable on a non-conducting surface, such as the inside of the Pod box or printout paper. Do not insert it into the blue foam. Most systems will work properly with just the Flex Cable installed. If your system does not, please call Tech Support for help.

### **Jumpers on the 286/386 Pod**

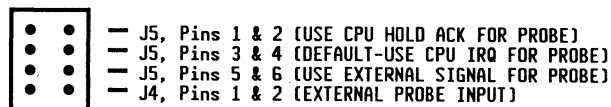
Under most conditions you won't need to change any settings other than J1 on the 286/386 Pod. The pins for the jumpers are numbered from left to right, top to bottom, starting at one. Be sure to hold the board so that the silkscreened jumper names are right-side up.



*Figure 7-6. Layout of the 286/386 Pod*

**J1 (286/386 Pod) - CPU SELECT.** The two pins of J1 must be connected when the Pod is used in an 80386 system. When the Pod is used in a 80286 system, the two pins must not be connected.

**J4 (286/386 Pod) - EXTERNAL PROBE INPUT.** When the Pod is held so that the PGA sockets are toward the bottom, J4 is the bottom two pins, with pin 1 on the left and pin 2 on the right. Pin 2 is used as ground and Pin 1 is used as an external probe input. It must be driven by a TTL-compatible signal. When J5 is set to select the external probe and the external signal is high, the word **Probe** appears in the hardware trace buffer.



*Figure 7-7. 286/386 Pod: J4 and J5 Pin Assignment*

**J5 (286/386 Pod) - PROBE SELECT.** When the Pod is held so that the PGA sockets are toward the bottom, J5 is the top six pins, just above J4. These pins are used to select the

---

source of the probe bit in the hardware trace buffer. The three valid positions are:

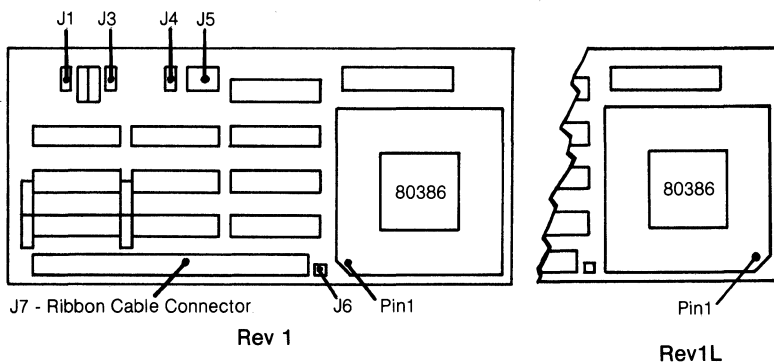
**Pins 1 and 2:** when these two pins are connected, the CPU Hold Acknowledge signal is used for the probe bit.

**Pins 3 and 4:** when these two pins are connected, the CPU IRQ line is used for the probe bit (default).

**Pins 5 and 6:** when these two pins are connected, the external probe input is used for the probe bit.

**J6 (286/386 Pod) - RESET/NMI CLIP.** This pin can be connected to the motherboard to allow resetting the CPU or generating an NMI via the Model IV Board.

### Jumpers on the 386 Pod



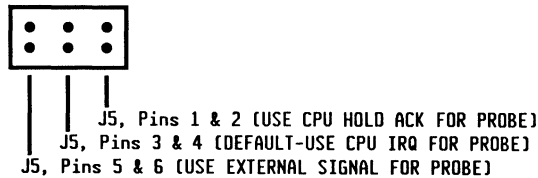
*Figure 7-8. Layout of the 386 Rev 1 Pod*

**J1 and J3 (386 Pod) - Always shunted.**

**J4 (386 Pod) - EXTERNAL PROBE INPUT.** Pin 1 is the one nearer the ribbon cable connector. Otherwise, this jumper is the same as on the 286/386 Pod.

**J5 (386 Pod) - PROBE SELECT.** Pin 1 is the one nearest the 80386 socket and on the inside row that is nearer to the ribbon cable connector. Otherwise this jumper is the same as on the 286/386 Pod.





*Figure 7-9. 386 Pod, Rev 1: J5 Pin Assignment*

**J6 (386 Pod) - RESET/NMI CLIP.** This jumper is the same as on the 286/386 Pod.

### **Jumpers on the Model IV Board**

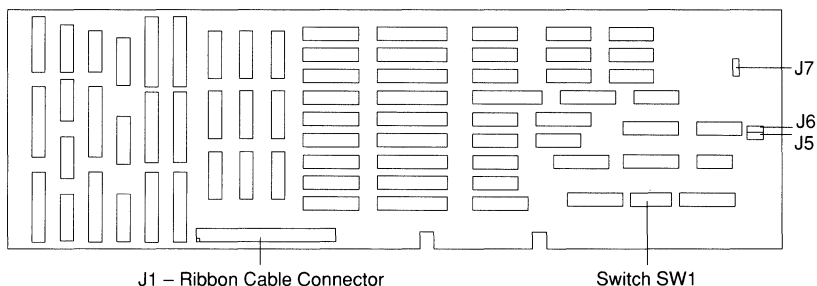
**J5 (Model IV Board) - PHONO JACK SELECTOR.** This jumper is used to select which signal is generated when the remote break-out switch connected to the phono jack is pressed. When the jumper shunts pins 1 and 2 (the two furthest away from the mounting bracket), an NMI is generated when the remote break-out switch is pressed (default). When the jumper shunts pins 2 and 3, pressing the remote break-out switch activates the reset signal.

**NOTE:** To actually generate a CPU reset, you must also set J7 (Model IV Board) and connect J6 (Pod) to an appropriate location in your system.

**J6 (Model IV Board) - PUSH BUTTON SELECTOR.** This jumper is used to select which signal is generated when the optional break-out switch on the board's mounting bracket is pressed. When the jumper shunts pins 1 and 2 (the two furthest away from the mounting bracket), an NMI is generated when the break-out switch is pressed (default). When the jumper shunts pins 2 and 3, pressing the break-out switch activates the reset signal.

**NOTE:** To actually generate a CPU reset, you must also set J7 (Model IV Board) and connect J6 (Pod) to an appropriate location in your system.

**J7 (Model IV Board) - EXTERNAL SIGNAL SELECT.** This jumper is used to control the signal source for J6 (Pod). When the jumper shunts pins 1 and 2 (the two pins



*Figure 7-10. Layout of the Model IV Board*

furthest from the gold fingers), the reset signal is selected. When the jumper shunts pins 2 and 3 (the two pins nearest to the gold fingers), the NMI signal is selected (default).

**NOTE:** To actually generate a CPU reset, you must also set J5 or J6 (Model IV board) and connect J6 (Pod) to an appropriate location in your system.

### **Model IV Installation Accessories**

The Periscope Company has available or in development socket extenders, spacers, and pods that can make installation of Model IV possible in machines it would not be possible in otherwise. If you have installation problems or would like to use your Model IV in additional machines, please contact Tech Support. There may be a solution to your problem now or in the near future. ♦

# Installing the Software



- Installation Options
- Alternate Start-up Methods

**T**his chapter describes how to install the Periscope software.

---

## INSTALLATION OPTIONS

To load Periscope, enter **PS** from the DOS prompt, followed by the desired installation options. Periscope needs approximately 124K of transient memory at load time.

If the Alt and Ctrl keys are pressed while **PS.COM** is loading, Periscope will terminate. This procedure can be used to keep Periscope from loading from **AUTOEXEC.BAT** (or any other batch file) without having to modify the batch file.

All installation options contain a forward slash (/) and a one- to three-character mnemonic, except as shown below. Some of the options include a number. If a number is used, it is always preceded by a colon (:) and the number is always in hexadecimal notation. Periscope can be installed multiple times per DOS session, but each install allocates more memory, unless the Model I board is used or the /Y installation option is used to remove the previous copy from memory.

The installation options are:

**? — Display help information about Periscope's installation options.**

**/2:px — Use Periscope in active remote mode.**

This option requires the use of the Periscope/Remote software, which is included in a separate package. See the documentation included with Periscope/Remote for more information.

**/25 — Use Periscope in split-screen mode on a VGA.**

This option is useful for debugging text-based applications on a VGA display. It sets the display to 50-line mode and uses the top 25 lines for your program and the bottom 25 lines for Periscope. This option cannot be used on an EGA or with graphics applications. For best results, use **ANSI.SYS** when using this option.

---

**/286 — Use Periscope Model IV in passive remote mode where the target system contains an 80286 CPU.**

To use this option, you must have the Model IV pod installed in another system, which uses an 80286. See Chapter 11 for more information.

**/386 — Use Periscope Model IV in passive remote mode where the target system contains an 80386 CPU.**

To use this option, you must have the Model IV pod installed in another system, which uses an 80386. See Chapter 11 for more information.

**/43 or /50 — Use the EGA 43-line mode or the VGA 50-line mode.**

When one of these options is used, Periscope supports a single monitor in the mode specified, presuming that the display is already in the specified mode when the resident portion of Periscope is activated. These options reserve 8KB of memory for Periscope's screen and 8KB for the application's screen. Dual-monitor support is not currently available for 43- or 50-line mode. Periscope will use 25 lines unless the screen was already in 43- or 50-line text mode. If Periscope is set to 43- or 50-line mode with screen swap off and the program's screen is set to 25-line mode, Periscope will revert to 25-line mode and turn screen swap back on.

**/A — Use an alternate monitor.**

This option indicates that you have both a monochrome and a color monitor attached to the system via separate display adapters. Periscope uses the monitor that is not currently active when the break-out switch is pressed or when a program is loaded with RUN. If this option is used, the /S option is ignored and no memory is reserved for the program's or Periscope's screen buffer. If Periscope is unable to initialize the second monitor, this option is ignored.

**/AD [:nn] — Use a Dual-VGA as an alternate monitor.**

---

This option indicates that you have a Dual-VGA made by Colorgraphic Communications Corporation in the system and want to use it as the alternate display adapter for Periscope. These boards are available in both ISA and MC versions.

In ISA busses, any built-in VGA must be disabled. The Dual-VGA card provides two separate display adapters, referred to as section A and section B, on the same card. Periscope will use the section that is not currently used by your program. A monochrome card may also be present in the system. For ISA systems, the option `/AD : nnn`, where `nnn` is the port number of the board may be used if the board is at an address other than 348H.

In MC systems, the Dual-VGA card is available with one or two sections. Periscope can use either section, while your program uses the built-in VGA only. To specify the section to be used by Periscope, you can use the option `/AD : A` or `/AD : B`. The default is section A (the lower connector).

In both types of systems, both displays must be color displays and only one Dual-VGA card is allowed. These restrictions are imposed by Periscope, not the Dual-VGA card itself. See the file `NOTES.TXT` for more information on Dual-VGA support.

**`/AK [:px]` — Use an alternate PC for Periscope's keyboard.**

This option uses an available serial port to communicate over a null-modem cable to another PC that is used as Periscope's keyboard. See the description of `/AV` below for more information.

**NOTE:** Starting with Version 5, the support for a dumb terminal has been replaced by alternate PC support.

**`/AV [:px]` — Use an alternate PC for Periscope's display.**

This option uses an available serial port to communicate over a null-modem cable to another PC that is used for

---

Periscope's screen display.

**NOTE:** Starting with Version 5, the support for a dumb terminal has been replaced by alternate PC support.

To use the /AK and/or /AV options, there are two requirements:

- The alternate PC must be running the program PSTERM.COM before PS.COM can be loaded on the debugging system. Please see the description of PSTERM in Chapter 10.
- The two systems must be connected with a null-modem cable that has Receive and Transmit crossed and has CTS and RTS crossed. See the specifications for the null-modem cable in the file NOTES.TXT.

The /AK and /AV options default to using COM port 1 at the fast (115.2K) speed. To use another port or a different speed, use the form /AK:px or /AV:px, where p is the port number (1 through 8) and x is the speed (S, M, or F for Slow (9,600), Medium (38,400), or Fast (115,200) respectively). Be sure that the speed used by PS.COM matches the speed used by PSTERM.COM on the alternate PC! For best results, use the Fast speed. If you encounter timeouts or other problems, try using Medium speed.

When the alternate video is used, the screen appears exactly as it does when Periscope is running on the same system. If the alternate keyboard is used, pressing any key on the alternate system will wake up Periscope on the local system. If you ever get a timeout message on the alternate system, avoid giving it the Fail reply since this can cause a system hang. For more information, see the description of PSTERM in Chapter 10.

**/B:nn** — Set the size of the software trace buffer to something other than 1KB.

This option is used when you want more than the 32 trace entries available from the default buffer size. The one- or two-digit hexadecimal number nn is the number of KB desired. The number may be from zero to 3FH K, allowing

---

a maximum of 2016 entries. Remember that the input is in hex! When the Model I board is used, this buffer defaults to 8K.

**NOTE:** The hardware trace buffer available with Periscope IV is separate from this software buffer. See Chapter 11.

**/C:nn — This option sets the color attribute for Periscope's display.**

The two digit number nn is from 1 to FFH.

Assuming that you want white on blue, use /C:1F. There are some illegal color combinations that Periscope won't allow. These include 0, 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.

To calculate the number you want, see the file NOTES.TXT or use the /C command when Periscope's screen is displayed.

**/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:n — Set the size of the source file buffer to something other than 2KB.**

This option is used to improve performance when the Ûassemble Source or View file commands are used. The one-digit hexadecimal number n is the number of KB desired. The number may be from zero to 8KB.

This number should usually be left at 2 unless you're debugging large files. Then make the buffer size one thirty-second of the size of the largest file. When the Model I board is



---

used, the source buffer defaults to 8KB.

## **/H — Install the on-line help and interrupt comments.**

If this option is specified, the files PSHELP.TXT and PSINT.DAT are loaded into RAM and will be available from Periscope. This option defaults to 'on' if either the menu system or a Model I board is used. The files must be in the current directory or must be able to be found via the Periscope path. To set the Periscope path, enter **SET PS=xxx** at the DOS prompt, where **xxx** is the desired path. 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.

The file PSINT.TXT is a normal ASCII file that is used to contain interrupt and I/O port comments. It is compiled to the file PSINT.DAT when CONFIG is run. The source file can be edited as needed to add or remove interrupt or I/O port information.

Each line in the file contains one of the following three formats:

- 1) An interrupt number, a space, a 2-byte function number (value of register AH), a space, and a description of up to 26 characters.
- 2) An interrupt number, a space, a 4-byte number (value of register AX), a space, and a description of up to 26 characters.
- 3) Two asterisks, a space, a 1- to 4-digit port number, a space, and a description of up to 26 characters.

Each line must end with a carriage return and line feed. The numbers are all in hex. If a function number is '.', the associated description is displayed for any value of AH or when the interrupt is not the current instruction. See the file PSINT.TXT for examples of each of the three formats.

Each line in the file PSINT.TXT uses 32 bytes in the file

---

PSINT.DAT. The maximum size of the file PSINT.DAT is 64K, so the maximum number of lines in the file PSINT.TXT is 2048.

To load only one of the files PSHELP.TXT or PSINT.TXT, rename the other file to some other name.

**/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 in Chapter 10. The two-digit hexadecimal number nn must be from 60H to FFH.

**/K:nn [nn]** — This option is used to mask hardware interrupt request (IRQ) lines when Periscope's screen is displayed.

The value of nn may be from zero to FFH. The first value corresponds to the first 8259. The optional second value corresponds to the second 8259 used on AT-class machines. The default value is zero, meaning that no IRQ lines are masked. A one bit in nn masks the corresponding IRQ line.

For example, to mask the timer (IRQ 0), use /K:1. To mask the keyboard (IRQ 1), use /K:2. (Don't do this unless you're using an alternate keyboard!) To mask IRQ 1 and IRQ 2, use /K:6 (bits 1 and 2 on).

**NOTE:** When using the /K installation option, you're protected from an interrupt while in Periscope, but if you use a Trace, Go, or Jump command, interrupts may happen unless the DI flag is set.

**/L:nnnn** — Load Periscope starting at the specified segment.

Normally, Periscope II and II-X are loaded as low as possible, i.e., just after DOS. If you're debugging non-DOS programs, this option can be used to place Periscope 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 2000H paragraphs. For example, if the PSP is C00H and the top of memory is A000H, the limits for this option would be C10H through 8000H.

**NOTE:** This option is ignored when Model I is used, since no memory external to the board is used.

**/M:nnnn — Set the protected memory segment to something other than D000H.**

Periscope I uses 32KB of system memory above the lower 640K but within the first megabyte. 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 5.

**NOTE:** The /M installation option is not available for Periscope I/MC, II or II-X.

**/N — Run the Periscope I software without using the protected memory.**

If you need to run the Model I software without the Periscope I board, use this option.

**NOTE:** The /N installation option is ignored for Periscope II or II-X.

**/P:nnn — Set the starting protected memory port to something other than 300H.**

Periscope I uses two consecutive ports and Periscope IV uses eight consecutive ports. 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 Chapters 5 and 7. 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.

**NOTE:** The /P installation option is not available for Periscope I/MC, II or II-X.

**/Q — Activate Periscope at ROM-scan time.**

This option requires the use of Periscope I. To use it, boot the system normally and install Periscope with the /Q installation option. Then use the QL or QB command to perform a long or normal boot. When the ROM scan of Periscope's protected memory occurs, Periscope's screen is displayed. From here, you can trace through the remainder of the boot process. Three caveats:

- The protected memory must be in the ROM-scan region (usually C800 to E000)
- Recent versions of DOS zap INT 1 and INT 3 while DOS is loading (be careful not to set a code breakpoint across this section of code.)
- After the QB or QL command is used, DOS use is turned off until another copy of Periscope is loaded.

**NOTE:** If you're using a system that uses shadow RAM, the video interrupt vector (INT 10) may be pointing to memory that is not valid at ROM-scan time. For example, on Compaq 386 systems, the VGA ROM at C000 may be remapped to faster RAM at E000. If this is the case, either disable the RAM shadowing, repoint the interrupt vector to C000 before using the QL or QB command to boot the system, or use the /V:10 installation option when you install Periscope.

**/R:nn — Set the size of the record definition table to something other than 1KB.**

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 KB desired. The number may be from zero to 20H KB. Remember that

---

the input is in hex! See the description of RS, which determines the size required for a DEF file and verifies the DEF file, in Chapter 10.

**/S:nn — Set the size of the program's screen buffer to something other than 4KB.**

This option is ignored if the /A, /AD, or /AV options are used. It is used when debugging programs that use the CGA, EGA, or VGA. It is only required for single-monitor systems where a 4KB screen buffer is too small. The one or two-digit hexadecimal number nn is the number of KB desired, from zero to 40H KB. If you need 16KB, enter /S:10. Remember that the input is in hex! The maximum size allowable is 64KB, using /S:40. Keep the number as small as possible, since each Trace or Go command has to copy this buffer twice. If you're using graphics modes that need more than 16KB on an EGA or VGA, a dual-monitor system is recommended for speed.

**/T:nnn — Set the size of the symbol table to something other than 1KB.**

This option is used when debugging programs with large MAP files and a large resultant symbol table. The one- to three-digit hexadecimal number nnn is the number of KB desired. The number may be from zero to 1FFH (511) KB. Remember that the input is in hex! See Chapter 10 for information on TS, which determines the symbol table size required and generates the PSS file.

The /T installation option can be overridden when you use RUN to load a program. The symbol space used by Periscope can be increased temporarily by allocating the needed space with the appropriate RUN option. See the description of RUN in Chapter 10 for details.

A second /T option can be used to set up two symbol tables. Then use the /1 and /2 commands to switch between the two symbol tables.

**NOTE:** When Model I is used, the default symbol table size is 80H (128) KB.

---

**/V:nn — Indicate a BIOS interrupt vector that is to be left alone while Periscope is active.**

Normally, each time Periscope's screen is displayed, it saves your program's interrupt vectors then temporarily resets the interrupt vectors it uses to their power-on default values. This is done to make Periscope as dependable as possible. When control is returned to the interrupted program, the interrupt vectors are restored to their prior values.

For example, assume your program changes INT 9. If you use /V:9, Periscope uses your program's vector. If your program crashes, it may take Periscope with it. If you don't use /V:9, Periscope saves your program's vector, swaps to a known good value (in BIOS) for its use, and then restores your program's vector on exit from Periscope.

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), 15H (cassette/scheduler), 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.

**NOTE:** 386MAX and other 386 control programs must have interrupt vector 15H left alone while Periscope is active. To automate this process on a 386 system, Periscope looks for a handle indicating that one of the following is present: 386MAX.SYS, QEMM.SYS, or QEXT.SYS. If one of these drivers is found, Periscope automatically generates a /V:15 installation option.

Here are some specific situations where you should use the /v option:

- 
- If you're using one of the keyboard translation programs (KEYBxx.COM), use /V:9 to keep the keyboard handler available from within Periscope.
  - If you're using Hercules graphics, run the public domain program HERC and use /V:9 to be able to switch screen modes from within Periscope.
  - If you're using a serial printer, run MODE and use /V:17 to be able to access your printer from within Periscope.

**/W — This option is used to set Periscope's windows.**

Please see the Option W command in Chapter 15 for the syntax and a complete description. (The window specification can be entered as an installation option when Periscope is run or as a command from within Periscope.)

**/Y — This option removes the current copy of Periscope from memory.**

It releases any DOS memory used by Periscope, and restores any vectors used by Periscope to their original values.

It should be used without any other options, and cannot be used if the /L installation option was used when Periscope was loaded.

**NOTE:** When using Model I, this option is not needed since Periscope can be run as many times as desired. Each copy of Periscope will overlay the previous copy in the board's protected memory, and will never consume any DOS memory.

**WARNING:** If another resident program has been installed after Periscope, this option can leave a black hole in the middle of memory.

**Examples.**

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/WD:8 RU: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 (starting at A000:0000) for the protected memory and use ports starting at 31C. Reserve 16KB to save the program's screen on a single-monitor system.

**PS/T:20/V:10/H** — Reserve 32KB 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 tables can be from zero KB to over 511KB. These buffers are located in normal RAM using the terminate-and-stay-resident function of DOS. No external buffers are used with the Model I board.

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. If you are not sure whether Periscope is installed, do not press the break-out switch. Execute RUN 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 the possible options. To use this method, enter **PS \*** at the DOS prompt. The screen shown in Figure 8-1 is displayed. (A second screen helps you customize the Periscope windows if you do not want to use the defaults.)

All information messages are suppressed when the installation screen is used. Only error messages are displayed.



---

Periscope Version 5.00P Copyright 1986-1990, The Periscope Company, Inc.

<p>25 - Use split 25/25-line mode on a VGA (y/n)? [N] 43 - Use 43 or 50-line mode on an EGA or VGA (y/n)? [N] A - Use an alternate monitor (y/n)? [N] B - Software trace buffer size [0-3FH kb] [01] C - Screen color attribute [1-FFH] [07] D - Restore original INT 13H for short boot (y/n)? [N] E - Source file buffer size [0-8 kb] [2] H - Install help and interrupt comments (y/n)? [N] I - User interrupt vector number [0-FFH] [00] K - Hardware interrupt [IRQ] masks [0-FFH] [00] [00] L - Load Periscope tables at segment [0000] M - Protected memory segment [xxxx-E000] [D000] N - Run without using protected memory (y/n)? [N] P - Base port [100-3FC] [300] Q - Activate Periscope when QB or QL command used (y/n)? [N] R - Record definition table size [0-20H kb] [01] S - Screen buffer size [0-40H kb] [04] T - Symbol table size [0-1FFH kb] [001] V - Leave BIOS interrupts alone while Periscope is active (y/n)? -- INT 08H [N] 09H [N] 10H [N] 15H [N] 16H [N] 17H [N] 1CH [N]</p>	<p>Key usage</p> <p>Enter - next field Home - first field End - last field Up - prior field Down - next field Esc - exit to DOS PgDn - next screen F9 - write response file &amp; install PS F10 - install PS</p>
---	---

Options not available using this screen: /2, /286, /386, /AD, /AK, /AV and /Y.  
Press PgDn for next screen...

---

*Figure 8-1. Periscope Installation Screen*

When F10 is used, the full-screen display generates a response file named PS that can later be used to start Periscope by entering PS @PS.

**NOTE:** The /2, /286, /386, /AD, /AK, /AV, and /Y installation options are not available from Periscope's full-screen display.

Normally, we recommend that you invoke Periscope from an AUTOEXEC.BAT file to ensure its presence each time the system is booted. If, however, 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 Periscope 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. ♦

# Using Periscope

- Symbols and Source Support
- Debugging at the Source Level
- Device Drivers
- PLINK Applications
- .RTLink Applications
- Microsoft Windows Applications
- Non-DOS and Pre-DOS Programs
- Debugging at ROM-Scan Time
- Hardware Interrupts
- Memory-Resident Programs
- Using an Alternate PC
- Using an EGA or a VGA
- PS/2 Machines
- Debugging Spawned (Child) Processes

**T**his chapter provides information for debugging in various environments supported by Periscope.



---

## SYMBOLS AND SOURCE SUPPORT

Symbols are the names you give to variables and procedures in your program. When you compile and link your program, the names are matched to their run-time memory locations. If you use a linker that outputs this name-and-location information to its MAP file or EXE file, Periscope is able to substitute the symbol names for the memory addresses, and can display source code, while you're debugging. This symbolic capability makes debugging much easier and faster.

As an example, assume that a program you're debugging calls a subroutine named `PRINTLINE`, and you want to go to the first call of this subroutine in your program. You enter `G PRINTLINE`. (`G` is the mnemonic for the Periscope Go command.) If symbols were not available, you'd have to know the address of the subroutine in order to get to it. When you disassemble this same program, the disassembly displays `PRINTLINE` wherever it is referenced in the program.

There are three major categories of symbols: public symbols, line-number symbols, and local symbols. **Public symbols** are global symbols that may reference code or data. **Line-number symbols** reference line numbers in your source program. Line-number symbols are the hook that makes source-level debugging possible. Both public and line-number symbols (but not local symbols) are available via the MAP file for most languages.

**Local symbols** include local code, local data, and stack data. Local code and local data symbols are names that are valid only within the scope of a particular module, i.e., the current code segment matches that of the module and the instruction pointer is within the range of the module. The locations corresponding to these symbols are fixed and may be accessed even when the module is not active, although the symbol name cannot be used when the module is not active. Stack data symbols have the same scope restrictions, plus even further restrictions on the range of the instruction pointer, since the stack must be set up before these values can be accessed.

For some languages, like most Microsoft and Borland

---

products, local symbols are available in the EXE file, and Periscope provides local symbol support via the EXE file. Periscope does not currently support typing, structures, or register variables.

**Notes on Periscope's Symbolic and Source-level Support.** Symbol names may be up to 32 characters. Any characters beyond the 32-character limit are discarded by Periscope.

A single source statement that spans two or more lines may generate only one line-number reference, causing Periscope to show only one line when displaying the source code.

Line numbers are formed by the module name, a '#', and the line number. For example, line number 12 in the module FTOC is called FTOC#12. When in the module, the short form #12 may be used.

To force Periscope to interpret a name as a symbol, use a period before the name. To force Periscope to interpret an item as not being a symbol, use a tilde (~) before the name.

For best results, turn code optimization off when using source-level debugging, since the line sequence can become quite scrambled when optimization is on.

To display the current symbols, press Alt-I. Unloaded symbols are shown in parentheses. These unloaded symbols can only be referenced by a BC or G command. To see all symbols for an address, enter /address and press Alt-I.

**Supported Compilers and Linkers.** Periscope works with programs written in any language. However, the amount of symbol and source-level support Periscope can provide to you varies depending on the compiler and linker you use. This is because Periscope normally gets symbol information from the MAP or EXE file output by the linker, and the linker gets symbol information from the OBJ files generated by the compiler.

**NOTE:** If your compiler and/or linker are not discussed below, please call Tech Support. We'll be happy to help you figure out how to get the

---

maximum symbol support for your programming environment.

**Borland's Assembler.** Use the `/zi` option.

**Borland's Compilers.** Use the `-v` compiler option to output public, line-number, and local symbols to the OBJ file.

**Microsoft's Assembler and Compilers.** Use the `/zi` compiler option to output public, line-number, and local symbols to the OBJ file.

**SLR's OPTASM.** Use the `/zi` option.

**Other Compilers.** Here's how you can find out what symbol information your compiler provides:

- Compile a test program.
- Run the IBM/Microsoft linker, specifying a MAP file and the `/LI` and `/M` options.
- Look at the MAP file. Any of the names in the 'Publics by Value' list or any of the line numbers at the end of the MAP file can be used as symbols.

**NOTE:** External references to other modules or sub-routines are always included in the MAP file. Line numbers indicate the beginning of a source line in memory and are required for Periscope source-level support.

**Borland's TLINK.** Use the `/M` and `/LI` options to output public and line-number symbols to the MAP file. Use the `/V` option to output public, line-number, and local symbols to the EXE file.

**Microsoft's LINK and LINK4.** Use the `/M` and `/LI` options to output public and line-number symbols to the MAP file. Use the `/CO` option to output public, line-number, and local symbols to the EXE file.

**SLR's OPTLINK.** Use the same options as with Microsoft's LINK.

---

**PocketSoft's .RTLink.** Specify a MAP file and use the **S**, **A**, and **L** options to output public and line-number symbols.

**Sage's PLINK.** Use SYMTABLE as a linker directive to output all possible symbols to the EXE file. There is no local symbol support with PLINK.

## DEBUGGING AT THE SOURCE LEVEL

To debug at the source level, you'll need to get line numbers into your symbol table by using the necessary compile and link options. You can verify that line numbers are in your symbol table by using the **/V** switch when you run TS.COM. This will display the count of various symbols, including line numbers. If no line symbols are indicated, recheck your compile and/or link options.

We suggest that you specify the full path name of the source file to the compiler. That way, the full path name will be passed on to the linker and Periscope so that Periscope can find your source file no matter what directory you debug your program from. If you are debugging on a system whose file structure is different than the system the program was compiled on, you can use the DOS APPEND program to pull together the source code from one or more directories.

When using RUN to load a high-level language program, you'll start at the very beginning of the program, which is usually an assembly prolog. To get to your source code, assuming a C program, enter **G \_MAIN**.

If you have problems displaying source code, check the following:

- Use the **/L** command to confirm the presence of line symbols, the availability of DOS, and the presence of a source file buffer. See the **/L** command in Chapter 15.
- You must be disassembling memory in an area where source line symbols exist (you should see some symbols of the form module name plus **'#'** plus the decimal line number).

---

## DEVICE DRIVERS

You can use the utility program `SYSLOAD.SYS` to load Periscope at `CONFIG.SYS` time. Then Periscope is available to debug your device driver. See the description of `SYSLOAD` in Chapter 10.

## PLINK APPLICATIONS

Using Periscope, you can set breakpoints in `PLINK` overlays. Be sure to set the breakpoints on program symbols only, however, since other locations are subject to being relocated without communication back to Periscope. The short form of the line number cannot be used to set breakpoints when `PLINK` overlays are used. Do not set a breakpoint at the symbol `$LDEX$`, since Periscope uses this location.

When loading a program generated with `PLINK`, `RUN` always uses the DOS `EXEC` function to load the `EXE` file.

At link time specify the `SYMTABLE` directive. When running `TS`, use the `/Q`, `/RP`, and `/RX` options. Also, set up the `MP` and `MX` aliases using `RS`. (See Chapter 10 for more information on `TS` and `RS`.) You may also interactively enter the aliases using the `EA` command.

## .RTLINK APPLICATIONS

You can set breakpoints in `.RTLlink` overlays using Periscope. You can get access to global and line-number symbols (no locals yet) for standard or overlaid programs. You'll need version 2.05 or later of `.RTLlink` and you'll need to use the `/P` option when you run `TS.COM`. When `RUN.COM` is used to load your program, it will use the DOS `EXEC` function. If you ever debug an `.RTLlink EXE` without a symbol table, be sure to use either the `/T` or `/X` option with `RUN.COM`.

When setting breakpoints in overlays, be sure to use symbol names instead of addresses, since Periscope needs to know the overlay to correctly set the breakpoint. Be careful to not



---

execute across an overlay reload using the Jump command, since this can cause Periscope to lose control. When debugging a program that uses reloads, set code breakpoints at the .RTLink vectors (see the .VEC file) to avoid problems using the Jump command.

If you're debugging programs that use RTLs, execute the following Periscope commands immediately after loading your program:

```
j;j;j;ls ds+10 xxxx;g $$main
```

where `xxxx` is replaced with your program's name. This sequence must be used to execute through the loading of the run-time library before the symbol table is loaded.

## MICROSOFT WINDOWS APPLICATIONS

Periscope supports Microsoft Windows, including the dynamic relocation of segments and communications to the debugging screen. For Microsoft Windows debugging, you'll need a dual-monitor system, with one display for Microsoft Windows (presumably an EGA or VGA) and one display for Periscope (typically an MDA). You may also use an alternate PC (see below).

**NOTE:** There is no support **yet** for other than real mode under Windows 3.0.

Use the `/A` installation option to tell Periscope to use the alternate screen. Do not use the `/V:9` or `/V:16` installation options when debugging Microsoft Windows applications, since Microsoft Windows' use and Periscope's use of these vectors conflict. Don't try to use PSKEY with Microsoft Windows, since Windows steals these vectors.

To debug a Microsoft Windows application, you'll need to create a PSS file using the `/w` or `/x` option with TS. (See Chapter 10.)

To enter Periscope from Microsoft Windows 1.03, press the SysReq key. If you have a 101-key keyboard that requires

---

pressing the Alt + SysReq keys, be sure to press the Alt key again on return from Periscope to 'release' it as far as Microsoft Windows is concerned. Then load your program's symbols using `LS CS name`. The load segment doesn't matter, since Microsoft Windows will fix it up at execution time. Set any breakpoints you need and enter `G` to return to Microsoft Windows. Then load your program to start debugging it.

If you're using Microsoft Windows 2.xx or 3.00, enter `RUN` with no arguments from the DOS prompt. Load your program's symbols using `LS CS name`. The load segment doesn't matter, since Microsoft Windows will fix it up at execution time. Set any breakpoints you need and enter `G` to return to the DOS prompt. Enter `WIN` to load Microsoft Windows. Then load your program to start debugging it. If you have the 101-key keyboard, there's no easy way to activate Periscope from the keyboard. Use the break-out switch to activate Periscope instead. Once in Periscope, if you need to get to a point where DOS is not busy, enter `G { 0 : 28 * 4 }` to get to the next invocation of `INT 28H`.

**NOTE:** A convenient breakpoint for Microsoft Windows applications is `WINMAIN`. To set the breakpoint, enter `BC WINMAIN`.

When using definition files with Microsoft Windows, you'll need to convert them to PSD files using `RS` and then load them dynamically using the `LD` command to avoid conflicts with Microsoft Windows DEF files. See Chapter 10 for more information on using `RS` and Chapter 15 for more information on the `LD` command.

When Microsoft Windows symbols are unloaded, the segment shows the relative segment number in parentheses when the Alt-I keys are pressed. Be sure not to load a symbol table while the target program is executing, since the symbols will be incorrectly marked as being unloaded.

Microsoft Windows can output messages to the Periscope screen when segments are allocated or moved. You can control the messages displayed using the `/M` command. See the description of this command in Chapter 15.

---

**NOTE:** When using Microsoft Windows, be sure to set breakpoints on program symbols only, since other locations are subject to being relocated without communication back to Periscope. Also, to be able to use Periscope with the debugging version of Windows, modify WIN.INI at the symbol ENABLESEGMENTCHECKSUM to disable checksumming. See the SDK for more information.

## NON-DOS AND PRE-DOS PROGRAMS

For non-DOS or pre-DOS programs, install Periscope normally, then 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, except when DOS 3.20 or later is used (see the `NOTES.TXT` file). If you are debugging non-DOS or pre-DOS programs, you can use the break-out switch after a short boot to get back into Periscope. If the timing is critical, embed an `INT 2` or `INT 3` in the code itself.

When performing a short boot, be aware that any DOS memory used by Periscope is no longer an extension of DOS and may be used by another program or garbled during the boot process. For best results, use the `/L` installation option with Models II and II-X to place the external tables in the middle of memory. If you have a Model I board in the system, all of Periscope's code and data are on the board and won't be corrupted by the short boot.

## DEBUGGING AT ROM-SCAN TIME

If you have a Model I board, you can have Periscope become active at ROM-scan time. See the description of the `/Q` installation option in Chapter 8. This method is better than the method described above that utilizes the `QS` command, since that command does not work in some systems.

---

## HARDWARE INTERRUPTS

To debug hardware interrupts most easily, the code should be in RAM so that you can set code breakpoints in the interrupt service code. Periscope's **T**, **GA**, or **GT** commands will not trace into hardware interrupts such as the Timer tick (IRQ 0) and the Keyboard (IRQ 1), so you must set a code breakpoint to be able to stop in the interrupt code. Once stopped, you can trace through the code as desired.

**NOTE:** For serious hardware interrupt work, you need Periscope IV to be able to set real-time breakpoints and to see just what happens when your code runs at full speed. See Chapter 11 for more information.

If Periscope must issue a non-specific End of Interrupt (EOI) to clear interrupts, a message of the form **EOI issued for IRQ x** is displayed, where **x** is zero for the timer and one for the keyboard, respectively. The EOI is one of the few things that Periscope cannot undo when control is returned to your program, so you may want to use a semaphore mechanism to keep your code from being re-entered while being debugged. On AT-class machines, Periscope can issue an EOI for IRQ 0 and 1 only, but on a PC-class machine, Periscope may issue an EOI for any IRQ level.

If you're debugging a keyboard or video handler, you may want to use an alternate PC for the keyboard or video. (See below.)

To prevent hardware interrupts from being executed while Periscope is active, use the **/K** installation option when you install Periscope.

## MEMORY-RESIDENT PROGRAMS

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 pro-

---

gram, the symbol table will remain available, ready for you at a push of the break-out switch! Source code will be available only when DOS is not busy, however. To keep DOS from being busy, use the WAITING.COM program described in Chapter 10.

## USING AN ALTERNATE PC

An alternate PC may be used for either an alternate screen or keyboard or both. The /AV installation option is used to select alternate video support, while the /AK option is used to select alternate keyboard support. See the descriptions of the /AK and /AV installation options in Chapter 8 for more information.

## USING AN EGA OR A VGA

Periscope supports the various EGA and VGA video modes. For single-monitor systems, a maximum of 64K of the screen buffer is saved and restored by Periscope. Generally, for graphics work, you'll want a dual-monitor system for best results.

Periscope has limited support of the EGA's 43-line mode and the VGA's 50-line mode. See the description of the /25, /43 and /50 installation options in Chapter 8.

## PS/2 MACHINES

Periscope supports dual VGAs on PS/2 systems using the Dual-VGA board made by Colorgraphic Communications. See the description of the /AD installation option in Chapter 8 for more details. You can also use another PC for an alternate keyboard, video, or both.

The PS/2 watchdog timer is supported. See the description of CONFIG in Chapter 4 for details.

The Periscope software works on all PS/2 machines. Periscope I/MC works on PS/2s with the Micro Channel bus. Periscope IV works on PS/2s with the 80286 CPU and ISA bus. Periscope IV works on PS/2s with either the 80286 or

---

80386 CPU and Micro Channel bus via the Periscope/Remote feature (which requires an ISA machine to act as host).

## DEBUGGING SPAWNED (CHILD) PROCESSES

If you want to debug a program that must be loaded by another program, you won't be able to use Periscope's program loader RUN.COM to get your symbols loaded. You can still use Periscope to debug your program with one minor change — simply embed an INT 2 (software NMI) or an INT 3 (code breakpoint) in your program, possibly activated by a command-line switch.

When your program executes the interrupt, it will activate Periscope. To load the symbols for the program, enter `LS $ <file>`, where `<file>` is the name of your program. This syntax assumes you're debugging an EXE file. If you are debugging a COM file, use `LS CS <file>`.

From here, you can set breakpoints in the spawned program and debug it. Also, for debugging multiple processes, be aware of the dual symbol table support provided by Periscope. Please see the description of the `/1` and `/2` commands in Chapter 15.

To automate the above process, you may wish to use the SYMLOAD utility. Please see Chapter 10 for more information on this program. ♦

# Periscope Utilities

- Clearing NMI (CLEARNMI)
- Your Program's Interrupts (INT)
- Setting up Hot Keys (PSKEY)
- Using an Alternate PC (PSTERM)
- Testing the Model I Board (PSTEST)
- Testing the Model IV Board (PS4TEST)
- Record and Alias Definitions (RS)
- Periscope's Program Loader (RUN)
- Ignoring DOS Memory Access (SKIP21)
- Loading Symbol Tables (SYMLOAD)
- Debugging Device Drivers (SYSLOAD)
- Generating Symbol Tables (TS)
- Customizing Periscope (USEREXIT)
- When DOS is Busy (WAITING)

**T**his chapter describes the capabilities of Periscope's utility programs and how and when to use them.



## CLEARING NMI (CLEARNMI)

Despite the name, the non-maskable interrupt (NMI) used by the break-out switch can be masked out. Since the NMI signal travels through two ports going from the expansion bus to the CPU, these ports can disable the break-out switch. The memory-resident utility program CLEARNMI attaches to the user timer interrupt (1CH) and clears these two ports once a second.

To use it, load CLEARNMI anytime, preferably from your AUTOEXEC.BAT file. There are three options: /Q should be used only if you configured Periscope to run on a PC- or XT-based machine with a 286 or later turbo card installed; /R causes CLEARNMI to refresh the NMI vector to point to Periscope once every 18 timer ticks (the program 'learns' Periscope's address and then refreshes INT 2 as needed); and /Y removes the current copy of CLEARNMI from memory.

## YOUR PROGRAM'S INTERRUPTS (INT)

This program is used to display, save, or compare interrupt vectors. The three usage modes are:

**INT <file> /W** - save the current interrupt vectors to a file.

**INT [<file>] [/D <lowint> <hiint>]** - display the previously-saved interrupt vectors from a file (if no filename is present, the current vectors are displayed). The optional numbers **lowint** and **hiint** indicate the range of vectors to be displayed.

**INT <file> /C** - compare a previously saved file with the current interrupt vectors.

To see the interrupt vectors used by a resident program, save the current vectors using the /W option, load the program in question, and then compare the current vectors with the saved vectors using the /C option.



---

For example, to see the interrupts used by `CLEARNMI`, enter `INT CLEAR/W` to save the current interrupt vectors. Load `CLEARNMI` from DOS. Enter `INT CLEAR/C` to compare the current interrupt vectors with the values saved in the file `CLEAR`. (You can also display the saved vectors using `INT CLEAR/D`.)

## SETTING UP HOT KEYS (PSKEY)

`PSKEY` is a memory-resident utility program that enables you to select your own hot-key combination to activate Periscope. This program can use interrupts 2 or 3, 5, 9 and 15H, depending on the command-line options.

The command-line options available are:

- `3` — Use INT 3 instead of INT 2 to activate Periscope
- `A` — Alt (combined with other shift keys)
- `C` — Ctrl (combined with other shift keys)
- `I` — Insert (combined with other shift keys; must be first key pressed)
- `L` — Left shift (combined with other shift keys)
- `P` — Use Shift-PrtSc to activate Periscope (via INT 5)
- `R` — Right shift (combined with other shift keys)
- `S` — Use SysReq to activate Periscope (via INT 15H)
- `/Y` — Remove the current copy of `PSKEY` from memory

For example, `PSKEY LR` would activate Periscope when the Left and Right shift keys are simultaneously pressed.

**NOTE:** If you have configured Periscope as Model II-X, the `3` option must be used. This is necessary because Model II-X does not support INT 2 (NMI). If the `3` option is not used with Periscope II-X, `PSKEY` displays Error 150, indicating that it was not able to activate Periscope via INT 2. Avoid configuring Periscope as Model II-X. Configure it as Model II if possible.

Avoid using the `P` or `S` options since Periscope comes up in BIOS, far away from your code. However, since the shift key combinations 'back-end' the keyboard interrupt, a

---

single Trace command puts you back to the code that was interrupted by the key press. The shift key combinations can be defined as needed to avoid conflicts with other memory-resident programs.

If PSKEY cannot find Periscope via interrupts 2 or 3 when the hot keys are pressed, the message `Error 150 - Int x does not point to Periscope!` is displayed. This usually means that you've mismatched PS and PSKEY. Since PSKEY is loaded into normal memory and is dependent on hardware interrupts being enabled, use the break-out switch for maximum dependability.

## USING AN ALTERNATE PC (PSTERM)

To use Periscope's alternate PC support, you'll need another PC with an available COM port and a null-modem cable similar to the ones provided with Brooklyn Bridge 3.0 and Laplink III. See the file NOTES.TXT for a description of the cable required.

First, load PSTERM.COM in the alternate system. The only option is `/2:px`, where `p` is the com port (1-8) and `x` is the speed (S = Slow, M = Medium, or F = Fast). You should try to use the fast (115,200 baud) speed if at all possible.

On the host system, you'll need to install Periscope using the `/AV` and/or `/AK` installation options after PSTERM is running on the alternate system. Be sure to specify the `:px` syntax as above if you use other than port 1 or the Fast speed. See the description of the `/AV` and `/AK` installation options for more information.

To terminate PSTERM on the alternate system, press the Alt and Ctrl keys simultaneously. If you terminate PSTERM after Periscope has been loaded on the host system, you'll need to reload PSTERM and then reload Periscope.

---

## TESTING THE MODEL I BOARD (PSTEST)

PSTEST is used to perform memory tests of the Periscope I protected memory. The options available are:

/0 -- Test Periscope I, rev 1 board (16K)  
/1 -- Test Periscope I, rev 2 board (56K)  
/2 -- (default) Test Periscope I, rev 3 board (512/1024K) or the Periscope I/MC board (512/2048K)  
/3 -- Test Periscope III board (64K)  
/A -- Analyze memory use from segment A000H to F000H  
/C:nnnn -- Run tests multiple times, where nnnn is a hex number  
/E -- Set error exit mode, activating Periscope if an error is found  
/M:nnnn -- Use protected memory segment other than D000H  
/N -- Set noisy mode, beeping after each error  
/P:nnn -- Use write-protect port other than 300H  
/S -- Set silent mode, showing only error messages  
/V -- Set verbose mode, placing each message on a separate line

**NOTE:** The /M and /P options are not used when testing Model I/MC.

Various memory tests are performed to verify the proper operation of the board. If no errors are found, the message **No errors detected** is displayed.

If you encounter errors, please check the following:

- If the base port has been changed from 300H, have you specified the correct port using the /P parameter?(not needed for Model I/MC)
- If the base memory address has been changed from D000H, have you specified the correct address using the /M parameter? (not needed for Model I/MC)
- If you get error 168, try adding wait states to the Model I, Rev 3 board using jumper J2.
- Are there any conflicts with the memory or port address you're using? Run PSTEST/A without the board in the system to check memory usage. To check I/O port usage, check the documentation for your system and



---

any add-in boards and see the port usage section in the file NOTES.TXT.

- Finally, if all else fails, try running the Model I, Rev 3 board in 8-bit mode by placing it in an 8-bit slot or by removing jumper J3.

## TESTING THE MODEL IV BOARD (PS4TEST)

The utility program PS4TEST is used to perform breakpoint and trace buffer tests on the Model IV Board.

The options available for PS4TEST are:

- /2 — Test board when board and pod are in different systems.
- /B — Test the break-out switch
- /C:nnnn — Run tests multiple times, where nnnn is a hex number
- /E — Set error exit mode, activating Periscope if an error is found
- /F — Run full trigger tests
- /N — Set noisy mode, beeping after each error
- /P:nnn — Use base I/O port other than 300H (must be evenly divisible by eight)
- /Q — The system has a PC or XT motherboard with an 80286 or later turbo card
- /S — Set silent mode, showing only error messages
- /V — Set verbose mode, placing each message on a separate line
- /W — Write the contents of the hardware trace buffer to PSBUF.DAT
- /WA — Copy the saved trace buffer from the diskette in A: to PSBUF.DAT
- /WB — Copy the saved trace buffer from the diskette in B: to PSBUF.DAT
- /X — Perform full extended memory tests (does not work on some systems!)

PS4TEST performs many different tests to validate the correct operation of a Model IV Board. If no errors occur, the message **No errors detected** is displayed. If an error occurs, a message is displayed and the next test is begun. If you get any errors, please check the following items before

---

calling Tech Support.

- Is the computer an 80286- or 80386-based system? As of this writing, Periscope IV cannot be used in 8088, 8086, 80186, 80386 SX, or 80486-based systems.
- Is the board being run faster than its rated speed? Check the speed rating of the board versus the CPU speed displayed by PS4TEST.
- Check the setting of the DIP switch. If it's been changed from the standard setting, be sure to specify the appropriate /P option when you install the Periscope software or run PS4TEST.
- Is the base port used by Periscope IV in use by any board other than a Model I, Rev 3 board? The port setting for the Model IV Board must be at an address that is evenly divisible by eight (300, 308, 310, etc.). The board uses eight consecutive ports starting at the base address.
- Confirm that all connections are fully seated. Pay special attention to the Flex Cable connections to the CPU and the motherboard as well as the Flex Cable connection to the Pod. When the Flex Cable is plugged into the Pod, you should hear a 'click' as the socket seats!

## RECORD AND ALIAS DEFINITIONS (RS)

RS is used to verify and size a record definition file. It reads a DEF file containing record and alias definitions and creates a Periscope definition (PSD) file. These definitions are loaded by RUN to provide support for Periscope's DR command and commands that use the aliases. To run this program, enter `RS filename` from the DOS prompt. The file extension is presumed to be DEF. Use the size shown by RS for the Periscope /R installation option. The PSD file can be loaded while in Periscope using the LD command.

**NOTE:** You must edit existing DEF files that use keyboard definitions F1-F10 to assign Ctrl-F1 through Ctrl-F10, since beginning with Version 5, F1 through F9 refer to key assignments for the function keys F1 through F9. (F10 is repre-



---

sented as F0.) Also, you must regenerate all PSD files for use with Version 5. There is no support for the old-style DEF or PSD files!

A section of the PS.DEF file is shown in Figure 10-1. It con-

---

```
\c1=k;dr cs:0 .psp;
\c2=dr cs:5c .fcb;
\FCB          ; File Control Block
Drive,b,1     ; Drive 0=default, 1=A, 2=B, etc.
File,b,8      ; File name
Ext,b,3       ; File extension
Block #,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
Res.,+,a     ; Reserved for DOS
Rec #,b,1    ; Current relative record number
Rel Rec #,d,4 ; Relative record number from beginning of file
```

---

*Figure 10-1. A Section of the PS.DEF File*

tains two alias definitions and a record definition. The third line of the file shown in Figure 10-1 starts a record definition named FCB.

The format for a record definition is:

- A backslash and the record name of up to 16 characters on one line
- One or more field definitions that contain the following separated by commas:
  - The field name of up to 10 characters;
  - the display type (any display format) plus an optional bracket or brace indicating a near or far pointer respectively;
  - the field length (total length in hex bytes);
  - optional comments preceded by a semi-colon (no preceding comma)

Each field definition is on a line by itself. If the field display type is long real (L), the length must be a multiple of eight. If the field display type is long integer (X), double word (D), or short real (S), the length must be a multiple of four. If the field display type is word (W), integer (I or Y), or number (N), 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. A field display type of + skips over the indicated number of bytes without displaying anything.

The record definition can reference the contents of near and far pointers. To use a field as a pointer, add a bracket ([]) for a near pointer or a brace {} for a far pointer immediately after the field display type. A near pointer always uses a word offset from the current segment and a far pointer always uses a double word segment:offset.

For example, the field definition `farptr,b{,10` uses the double word at the current location as a far pointer and displays the target of that pointer in 'b'yte format for 10H bytes, with a name of `farptr`. Similarly, the field definition `nearptr,d[,4` uses the word at the current location as a near pointer and displays the target of that pointer in 'd'ouble word format for 4 bytes, with a name of `nearptr`.

Note that each pointer must be individually described in the record definition and that only one level of indirection is supported. To remind yourself that the field is a pointer, you may wish to start the field name with a bracket or brace.

The other type of definition in a DEF file is the alias. An alias is a two-character mnemonic that represents a string of up to 64 characters. An alias may be entered as a line in a DEF file or may be entered using the EA command.

Aliases may be used in various ways:

- To assign commands to function keys (F1 through F10, Ctrl-F1 through Ctrl-F8, Alt-F1 through Alt-F10, and Shift-F1 through Shift-F10)
- To store special strings for use by Periscope (see the MP, MX, and X0 through X3 aliases)
- To assign character strings or commands to any alias that is not reserved by Periscope and then execute the alias by typing ^ and the two-character alias name. Aliases can be chained (not nested) by embedding a ^xx at the end of one alias to start up the next one. Watch out for infinite loops, such as ^Y1 chaining to ^Y2, which chains to ^Y1.

---

The first two lines of the file in Figure 10-1 contain alias definitions for function keys Ctrl-F1 and Ctrl-F2. While in Periscope, these aliases may be executed by pressing Ctrl-F1 or Ctrl-F2.

The format for an alias definition in a DEF file is:

- A backslash
- The two-character alias name
- An equal sign
- The one to 64-character string to be assigned to the alias

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. There may be up to 128 alias definitions in a DEF file.

There are various aliases currently reserved by Periscope. The reserved aliases are:

**MP** - The module path name for source-level debugging. If used, this alias should be the complete drive and path name, ending in a backslash.

**MX** - The module extension for source-level debugging. If used, this alias should be the file extension, starting with a period.

**NOTE:** Different linkers put different information in the MAP and EXE files, so the **MP** and **MX** aliases may or may not be needed to construct the full source file name. Periscope concatenates the path, name, and extension to get the module's full file name. If Periscope prompts you for a source file name, press Alt-C to see the file name Periscope tried to use. You can edit the file name and press return to try the file again. To fix the problem permanently, set the **MP** or **MX** aliases by using the **EA** command or by placing them in the DEF file for the program. If the path is missing or incorrect, use the **MP** alias to automate source-level debugging of



---

the program. Similarly, if the file extension is missing or incorrect, use the **MX** alias. Note that the path name used with the **MP** alias should end in a backslash.

**X0** — The commands executed when RUN transfers control to Periscope on entry to the program.

**X1** — The commands executed on entry to Periscope each time control is transferred from your program to Periscope.

**X2** — The commands executed after each Periscope command.

**X3** — The commands executed on exit from Periscope each time Periscope transfers control to your program.

**C0** — The commands to restore the last Periscope window settings (Ctrl-F10).

**C9** — The commands to restore the original (default) Periscope window settings (Ctrl-F9).

**Fx** — Defines the contents of the menu bar. Used with **xxKEYS.PSD**. See **NOTES.TXT**.

These are the aliases you can custom-define in the DEF file:

**A1 through A9 and A0** — Assign string to Alt-F1 through Alt-F10.

**C1 through C8** — Assign string to Ctrl-F1 through Ctrl-F8. (Ctrl-F9 and Ctrl-F10 are reserved by Periscope.)

**F1 through F9 and F0** — Assign string to F1 through F10.

**S1 through S9 and S0** — Assign string to Shift-F1 through Shift-F10.

See also Chapter 12.

## PERISCOPE'S PROGRAM LOADER (RUN)

The program RUN is Periscope's program loader. It can load data files, COM and EXE files, or no file at all.

RUN is started by entering

```
RUN [/2] [/T|/X[:nnn]] [<file>] [<command line>]
```

at the DOS prompt, where <file> is the path, file name,

---

and extension (EXE, COM or other) of the file to be loaded. If the file extension is omitted, COM, then EXE, then no extension is presumed.

When RUN is entered with no arguments, no file is loaded. The first instruction is set to INT 20H, the DOS return, to prevent accidental execution of meaningless data. This technique is useful for clearing and initializing Periscope, since RUN performs these tasks when it is executed:

- Resets Periscope's display address to the PSP segment at offset 100H.
- Clears some of Periscope's tables, including the software trace 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 current program being debugged.
- If code breakpoints are set, they are reversed out so as not to leave an INT 3 in the code.
- Clears the breakpoints and watch variables for a program if the date/time of the program is different than the last program so that one program's breakpoints/watch variables won't be used on another program.
- Clears the display windows to the standard values.

Loading a data file with RUN enables you to patch the file. If a data file is loaded, be sure to use the QR command to quit Periscope and return to DOS. Using the QC or G commands would have unpredictable results.

The command-line entered after the filename when RUN is started is the same one used when the program is started from DOS. RUN adjusts the FCBs and command line in the PSP to look like the target program had been started directly from DOS.

If an executable program (COM file or EXE file) is being loaded, RUN loads the related symbol table if a PSS or MAP file is found and the related record definition table if a PSD or DEF file is found.

---

To locate the symbol file, RUN first searches the specified directory for a file of the form `filename.PSS`. If this file is found, it is used to load the program's symbols. If the PSS file is not found or if the date of the PSS file precedes the date of the executable file, the directory is searched for a file of the form `filename.MAP`. If found, the program TS is executed to generate a PSS file.

**NOTE:** If TS is executed by RUN, a standard MAP file is presumed, i.e. no command-line options are set when TS is executed. If the Periscope path (`SET PS=`) is set, it is used to find `TS.COM`. Otherwise, the DOS path is used. If you've got a copy of Peter Norton's `TS.EXE` in your path, you'll need to set the Periscope path to avoid executing the wrong TS.

If no symbols are available or the symbols will not fit in the allocated space, the symbol table is cleared. When source lines are found in the PSS file, RUN sets Periscope for Source-only mode. If no source lines are found, the default is for mixed (Both) mode.

To locate the record definition file, RUN first searches the specified directory for a file of the form `filename.PSD`. If this file is not found, the directory is then searched for a file of the form `filename.DEF`. If it is not found, RUN then searches the Periscope directory, which you can set by entering `SET PS=xxx`, where `xxx` is the path required to find the file `PS.PSD`. If not found, RUN searches for `PS.DEF`. The PSD file or DEF file is used to load Periscope's record and alias definition tables. If a definition file is not found, the record and alias definitions are cleared. If an error is found in the DEF file, the record definition table will be partially loaded.

Once the symbol and definition tables are loaded, RUN 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. The register pair `BX:CX` is 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 (unless the /T or /X option is used), this area shows RUN 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, use the /T or /X options described below.

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, alias, 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 the G command with no breakpoints set, use the QC command to quit Periscope and continue execution, or use the QR command to quit Periscope 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.

If you're using two symbol tables, you can have RUN load the program's symbols into the second symbol table using RUN /2 <file>. If this option is not used, the symbols are loaded into the first symbol table. This option can only be used if two /T:nnn installation options were used when Periscope was installed.

The symbol space used by Periscope can be temporarily increased by two RUN options, /X[:nnn] and /T. When you specify either of these options, RUN uses the DOS EXEC function to load your program.

Temporary symbol space is allocated using the /X:nnn option immediately after RUN, where the optional nnn is from 1 to 1FFH (511) KB. For example, if the symbol size was set for 4KB, and the program TEST needs 8KB, use

---

**RUN /X:8 TEST.EXE** to allocate the temporary symbol table and load the program with the DOS EXEC function.

The **/T** option used immediately after **RUN** allocates the exact amount of symbol space required by the program, with at most 1KB to add symbols dynamically. It also uses the DOS EXEC function to load the program.

When using Periscope I, avoid the **/T** and **/X:nn** options. Use just the **/X** option, which uses the DOS EXEC function but does not allocate any temporary symbol space.

**NOTE:** The **/X[:nn]** option or **/T** option should be used if you experience problems using **RUN**. They both use the DOS EXEC function, so your program is loaded approximately 8KB higher in memory than otherwise, but the program name in the environment space is correct. To use the EXEC function, add **/T** or **/X[:nnn]** immediately after **RUN**. For example, enter **RUN/T FTOC**. If an EXEC error occurs, error 182 is displayed. This error usually indicates a bad path name.

Programs linked with **PLINK** or **.RTLink** must be loaded with the **/T** or **/X** option since these programs look for their name in the environment space. Periscope handles this automatically if a symbol file is used.

If **RUN** detects that the **NMI** vector has been stolen from Periscope, it displays Error 170. This usually indicates that your display adapter has stolen **INT 2**.

## **IGNORING DOS MEMORY ACCESS (SKIP21)**

A common problem in many C programs is the corruption of low memory due to uninitialized pointers. Periscope Model IV has the ability to watch for writes to low memory in real-time, but since DOS (and other programs) can legitimately write to this area, false hits can occur. This memory-resident program is used to disarm a Periscope IV board while **INT 21H** services are in use. Since **SKIP21**

---

does not disarm other interrupts, minimize the use of memory-resident programs when SKIP21 is in use. For best results, a Periscope Model I board should be used in conjunction with the Model IV board. Follow the instructions below exactly for proper operation.

**Step 1** — Install Periscope.

**Step 2** — Load SKIP21 from the DOS prompt. The options available are:

- /3 — A Model III board is in use
- /4 — A Model IV board is in use
- /I:nn — Specify interrupt to be used (default is 21H)
- /P:nnn — Set the base port for the Periscope board (default 300H)
- /Y — Remove SKIP21 from memory

**Step 3** — Load your program with RUN.

**Step 4** — Set hardware breakpoints on writes to the interrupt vector table (0:0 to 0:3FF) and from the end of the BIOS data area (0:500) to the beginning of your program (use the PSP segment displayed by RUN). Note that this avoids the BIOS data area (0:400 to 0:4FF), since this area is constantly being updated by various BIOS services.

**NOTE:** If you're not using a Model I board, you'll need to split the breakpoints up to avoid the region in DOS memory used by the Periscope software.

**Step 5** — Enter a code breakpoint at the termination point of your program. This breakpoint is very important, since DOS calls that do not return control to their caller cause SKIP21 to lose control.

**Step 6** — Enter GH to execute your program with the hardware breakpoints enabled. If an INT 21 service writes to one of the hardware breakpoint regions, it is ignored, but if your program writes to one of these areas, Periscope's screen is displayed.

**Step 7** — When you get to the end of the program, enter G

---

(not GH) to continue execution.

**Step 8** — Restart the program with RUN or re-enter the full SKIP21 command-line plus /Y to remove SKIP21 from memory.

## LOADING SYMBOL TABLES (SYMLOAD)

This program is used to load Periscope's symbol tables from within your program. This approach can be used when your program manages overlays or is not loaded by RUN. The LS command may also be used to load symbols on the fly.

SYMLOAD is a memory-resident routine that is run once per DOS session. The /Y option removes the current copy from memory. SYMLOAD 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 not be busy for SYMLOAD to work.

To install SYMLOAD, enter `SYMLOAD /I:nn`, where nn is the interrupt number to be used to access SYMLOAD. 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.

On return, register **AH** contains the status of the operation.  
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
- 5 - Logical error in symbol table

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 symbols whose segment was already in the range of **F000H** to **FFFFH**.

Register **AL** is used to return additional error information if a read error occurred.

## DEBUGGING DEVICE DRIVERS (SYSLOAD)

**SYSLOAD.SYS** is a utility program written by Bob Smith (386MAX's developer). Bob has licensed us to distribute this powerful program with Periscope. It allows you to load any of the **COM** programs in the Periscope package at **CONFIG.SYS** time as a device driver. By being able to load **PS** as a device driver, you can greatly ease the debugging of your own device drivers.

To use **SYSLOAD**, place a line of the form  
**device=sysload.sys arguments** in the **CONFIG.SYS** file. The **arguments** field may be one or more of the following:

**/I** is an optional argument that generates an **INT 3** to activate Periscope just before jumping to the specified program.

**/Q** is an optional argument that sets quiet mode and displays serious error messages only. This option is useful



---

when loading a transient program.

`/P=c:filename.ext arg1 arg2 ...` is required and must appear last. It specifies the drive, path, and file name of the program to be loaded, followed by the arguments the program needs. Be sure to specify the full file extension although only COM programs can be handled at present. Note that the part of DOS which loads device drivers also converts all characters to upper case. Thus case sensitive arguments cannot be passed to the program.

To load Periscope via SYSLOAD.SYS, use a line similar to `device=sysload.sys /p=c:\peri\ps.com/a` in CONFIG.SYS. Follow this line with the invocation of your device driver. Embed an INT 2 or an INT 3 in the strategy and/or interrupt entry points of the driver. When the driver is executed, Periscope's screen is displayed. You can then use the LS command to load the driver's symbols and begin debugging the driver.

## GENERATING SYMBOL TABLES (TS)

The Periscope program TS verifies and sizes MAP or other symbol files, and generates a Periscope symbol file with an extension of PSS. To run TS, enter `TS progname` from the DOS prompt. Unless you specify options that indicate otherwise (see below), TS assumes a file extension of MAP and assumes that the MAP file is in the same format as the sample file FTOC.MAP. For help when running TS, enter `TS ?`. TS needs approximately 180K of work space plus the size of the PSS file that is generated.

The options available are:

`/A xxxx` — Add a variable-length prefix to symbols.

`/B` — Read an EXE file as produced by Borland's TLINK

Periscope supports local symbols for most Borland products when compiled and linked with the appropriate options. You must run TS with the `/B` option to generate a PSS file that includes local as well as public and line number symbols.

---

Periscope does not currently support typing, structures, or register variables. You may need to use the `-r-` option with Turbo C to force register variables off if you have problems accessing all local variables. Also, watch out for libraries with extended dictionaries. You may need to rebuild the library without the `/e` switch.

As of this writing, enhancements to support the latest version of the Borland EXE format are in progress. Please check the READ.ME file on the distribution disk or call Periscope for the latest information.

`/C` — Read a DeSmet MAP file as produced by CWare's C compiler.

`/D` — Read a LINK86 SYM file as produced by Digital Research's LINK86.

`/E` — Read a CodeView EXE file as produced by Microsoft's LINK.

Local symbols are available for Microsoft products when compiled and linked with the appropriate options. You must run TS with the `/E` option to generate a PSS file that includes local as well as public and line number symbols.

**NOTE:** When using Microsoft's MASM with source code that uses the assembly `INCLUDE` statement, the CodeView EXE files contain incorrect information for the included files. For best results, you should turn off the generation of line numbers for the affected files or merge the included code inline. If it is not possible to do either of the above, try using the TS `/LD` option described below.

If you have problems generating symbols from an EXE file, try loading the program under CodeView and then use the `x` command to display symbols. If you get an error, call Microsoft. If all else fails, try using the MAP file. You won't have any local symbols, but at least you can use an editor to fix any problems.

---

**/F $x$**  — Filter leading character  $x$  from public symbols.

For example, Microsoft C uses leading underscores on public symbols. If you use **/F\_**, TS will remove any leading underscores from the symbols.

**NOTE:** A different filter function enables you to limit the size of your symbol table when you are only debugging a few modules in a large program. To use this filter function, create an ASCII text file with an extension of .PSF and one statement per line in the format **X=NAME**, where  $X$  is **L** for line number or **P** for public symbol. **NAME** is the name of the module or public symbol. The match is made on as many characters as are entered as **NAME**, and the match is case-sensitive. For example, the line **P=A** would filter out all public, local data, or local code (but not stack-based) symbols starting with **A**, while the line **L=FTOC** would filter out all lines in the module **FTOC** or **FTOC????**, where **????** may be any characters.

If you have some symbols that you don't want included in the symbol table, edit the MAP file and delete the desired symbols or insert braces as desired to turn off symbol generation. A left brace (**{**) turns symbol generation off, and a right brace (**}**) turns it back on. Put the braces in the beginning of existing lines. Do not add any new lines to the file! Be careful when saving the MAP file. Don't let any TABs or high-bit characters into the file.

**/LA** — Accept line numbers that are out of sequence. Use this option for programs produced by Borland's C compiler and other optimized code.

**/LD** — Discard all line numbers that are out of sequence. Use this option to discard all lines for a module after a discontinuity, such as CodeView information for an ASM program that uses includes.

**/P** — Read an .RTLink MAP file as produced by

---

PocketSoft's .RTLink. This option reads public and line-number symbols from the .RTLink MAP file. No local symbol support is available for .RTLink as of this writing.

**/Q** — Read a PLINK EXE file as produced by Sage's PLINK. This option reads public and line-number symbols from the PLINK EXE file. No local symbol support is available for PLINK as of this writing.

If static symbols are found with embedded brackets in a PLINK EXE file, TS converts the brackets to underscores. For example, FOO[BAR.C] becomes FOO\_BAR.C\_, unless the **/S** option is used, in which case the static symbols are not output into the PSS file.

**/RP** — Remove path from line-number records.

**/RX** — Remove file extension from line-number records.

The **/RP** and **/RX** options are intended for use with PLINK only, since it is inconsistent in the filenames it generates for line-number symbols. When using these options, be sure to use the **MP** and **MX** aliases to provide the needed path and extension.

**/S** — Filter static variables from a PLINK EXE file.

**/V** — Verbose mode: show statistics on the number and type of symbols.

**/W** — Read a Microsoft Windows MAP file as produced by Microsoft's LINK4.

The **/W** option does not provide any local symbol support.

**/X** — Read Microsoft Windows MAP and EXE files as produced by Microsoft's LINK4.

The **/X** option uses both the MAP and EXE files to provide access to local symbols for Windows. Use of C 5.x or later and Windows 2.x or later is required. When running LINK4, specify a MAP file and the **/LI/M/CO** options.

---

## CUSTOMIZING PERISCOPE (USEREXIT)

The USEREXIT 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 Periscope is installed. Also, the Periscope 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** — Points to Periscope's data area (see the file 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.

**ES:BX** — Points to a user service routine in Periscope. This routine is accessed via a far call. Register AH is used to indicate the function desired. When using this routine, all registers are preserved. The functions available are (see the sample program USEREXIT.ASM for details):

- Display nul-terminated string on the screen using Periscope's display handler
- Get command line using Periscope's keyboard handler
- Search symbol table using ASCIIZ string
- Search symbol table using an address

**NOTE:** Your suggestions for additional routines are welcome.



---

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 any screen output is done via the user service routine described above.

Do not attempt to perform DOS functions from user-written code as DOS may be busy! 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, run the program from DOS. Then install Periscope 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 not busy. Try `/U 9` as an example of a user exit modifying the command line. If you have a numeric co-processor, try `/U 87` to display the numeric processor status. To display the BASIC-style string `A$`, enter `/U D A$`. To display DOS's memory allocation, enter `/U 88`.

The user exit `/U A` sets a range of values for the CS register to be used by the user breakpoint `BU 2`. The range must be entered as a segment:offset pair, with the colon. To enable the user breakpoint, enter `BU 2` and then `GA`, `GM`, or `GT` as needed.

USEREXIT reflects these additional reserved user exits:

- **F0** — invoked after each command is entered, so that your code can modify the command line as needed.
- **F1** — invoked before a short boot (QS), so that your code can perform any required cleanup.
- **F2** — invoked before a normal boot (QB).

- 
- **F3** — invoked before a long boot (QL).

To debug user exits, do the following:

- Load Periscope normally, without the `/I:nn` installation option
- Load your user exit code.
- Enter `RUN/T PS /X/X/I:nn` to load a test copy of Periscope, adding any installation options you like at the end.
- Set a code breakpoint on the user exit using `BC {0:nn*4}`, where `nn` is the interrupt number.
- Use `G` to get to the test copy of Periscope. Note that the prompt is an asterisk.
- Enter `/U nn` to activate your user exit. Execution will stop in the real Periscope at the start of the routine.
- Debug your code.
- When done, use `QC` at the asterisk prompt to terminate the test copy of Periscope.

## WHEN DOS IS BUSY (WAITING)

If you're debugging TSRs that do not hook INT 21, a simple program running in the foreground can keep DOS available so you can use source-level debugging while your program is running. The program `WAITING.COM` continuously calls the BIOS keyboard handler while looking for a Ctrl-C. When a Ctrl-C is pressed, control is returned to DOS.

Use `RUN` to load your program and set whatever breakpoints you desire. When the DOS prompt is displayed, run `WAITING`. Then when debugging your program, DOS won't be busy. ♦







# Using Model IV

- **Model IV Capabilities**
- **Model IV Compatibility and Caveats**
- **Model IV Hardware Breakpoint Examples**
- **Model IV in Passive Remote Mode**

**P**eriscope IV is a powerful hardware-assisted debugger for use in 80286- and 80386-based (more coming!) personal computers. Its commands are a superset of the normal Periscope commands, so it provides both the capabilities of the other models of Periscope plus its own special hardware capabilities. This chapter summarizes Model IV's capabilities along with compatibility issues and caveats, provides examples of hardware breakpoints, and describes how to use Model IV in passive remote mode.



---

## MODEL IV CAPABILITIES

Periscope IV monitors the CPU, watching for user-specified breakpoints to occur while the test program is running at full speed. These breakpoints can be set on memory reads and writes, code prefetch, interrupt acknowledge, CPU halt, I/O port reads and writes, and data values. They may be further qualified by a pass counter. When the board detects that a breakpoint has been reached, it generates an NMI, which activates the resident Periscope software.

Periscope IV does not contain any write-protected memory. This memory is available, however, by using a Plus (Model I, Rev 3) board with the Periscope IV hardware. The Model I board keeps all debugging information out of the lower 640K of DOS memory. It can run with either 512K or 1MB of memory and has just a 32K footprint in the first megabyte of system memory, above the lower 640K.

The Periscope IV hardware provides two major functions:

- Hardware breakpoints
- Real-time trace buffer

**Hardware breakpoints** include:

- Memory access
- I/O port access
- Data values and/or data bit mask
- Pass counter
- Sequential triggers
- Selective capture

**Memory breakpoints** can be set on up to eight ranges in the first 16 megabytes of system memory. The options available for memory breakpoints include memory read, memory write, code prefetch, interrupt acknowledge, and processor halt. Since Model IV views the world from the perspective of the CPU, it has no access to DMA signals.

**Port breakpoints** can be set on up to eight ranges from zero to FFFFH. The options available for port breakpoints include I/O read and I/O write.

---

**Data breakpoints** can be set on byte, word, three-byte, and doubleword values. Data breakpoints can be used by themselves or to qualify a memory or port breakpoint. Up to eight data breakpoints can be set on byte ranges; specific word values; specific three-byte values, or specific doubleword values. A bit mask may also be used to indicate desired data values. Bit values of zero, one, or 'don't care' are all supported.

The **pass counter** is used to interrupt the executing program after the indicated number of breakpoints has occurred. The default pass count is one, which causes an interrupt on the first breakpoint. The value of the pass counter may be from one to FFFH. If the memory, port and/or data values found on the current CPU cycle would cause a breakpoint, the internal pass counter is decremented. When the pass counter reaches zero, the Periscope screen is displayed.

**Sequential triggers** are used to watch for a specific sequence of events in real-time. Periscope IV has an eight-state state machine that lets you construct up to seven levels of sequential triggers of arbitrary complexity. For example, you can set a trigger on the writing of a variable from a specific subroutine after data has been output to the printer.

**Selective capture** lets you control the type of information placed in the hardware trace buffer. Since the buffer is 2K CPU events deep, selective capturing can increase the effective depth of the buffer significantly.

Using the breakpoints described above, it is possible to trap complex events, such as when the variable BAR is written only while the routine FOO is executing.

Since some conditions, such as register values, are not visible from the CPU in real time, we've implemented a **hardware/software combination breakpoint** in Periscope IV. Using this capability, you can execute at full speed until a hardware breakpoint is reached, then drop into software to evaluate a condition. If the software condition gives a 'hit', Periscope's screen is displayed, otherwise full speed execution continues. See the description of the GM command



---

for more information.

If your program is overwriting low memory, setting memory breakpoints can be tricky, since DOS and other programs can be legitimately writing into the specified range. The utility SKIP21 filters out the effects of DOS's memory writes. See Chapter 10 for more information on SKIP21.

**The real-time trace buffer** is a circular buffer that captures up to 2047 (2K-1) CPU events. Each CPU event 'record' is 80 bits wide. It includes the 32-bit address; 32-bit data; 4-bit byte enable signal; 8-bit CPU cycle count; 3-bit status (interrupt acknowledge, I/O read, I/O write, code prefetch, CPU halt, memory read, or memory write); and a probe bit. The probe bit may be tied to an external probe input, the CPU IRQ signal (default), or the CPU Hold Acknowledge signal, depending on jumper settings on the pod.

The buffer can be displayed in three formats: a raw dump that shows the address, data, status, and cycle count information for each trace buffer record; a disassembly mode that uses the prefetch cycles found in the buffer to show only instructions; and a combination of the above two formats, which shows a mix of instructions and the data accesses performed by the instruction stream. Additionally, there's a powerful set of commands to search for items in the trace buffer and to control the amount and type of information displayed.

Once the Periscope software is loaded, the buffer is continuously updated while your program is running, even if you're not debugging. This means that you can press the break-out switch any time to see what's been happening. Also, if you should ever get an exception interrupt, you can look in the trace buffer to see what led up to the exception.

A breakpoint option is available to stop program execution when the trace buffer is filled, allowing you to examine the execution flow of a program starting at a known point and stopping after 2K CPU events have occurred. The selective capture capability lets you capture just trigger events in the trace buffer. This is useful when you need to capture multiple widely-spaced events in real-time.

---

When each trace buffer entry is written, the number of CPU cycles since the last trace buffer entry was written is saved in the 8-bit cycle count field. Under normal conditions, the cycle count information will rarely overflow. If the cycle count does overflow (such as when selective capture is in use) it is possible to force the generation of cycle count overflow records into the trace buffer, allowing you to count CPU cycles between widely-spaced events. The CPU cycle count can be easily converted to time using the CPU speed (see the table in Figure 15-5).

The trace buffer can be set to show the events surrounding a breakpoint in three different ways:

- up to 2K events before the breakpoint;
- up to 1K events before the breakpoint and 1K events after the breakpoint; or
- 2K events after the breakpoint.

For real-time applications, these capabilities are extremely valuable, since you can see how your program got where it did plus where it went afterwards, all in real-time!

## MODEL IV COMPATIBILITY AND CAVEATS

**Compatibility.** Periscope IV will work on most any 80286 or 80386 machine running up to 25MHz (80486 and 33MHz support are in development!) so long as it can be physically installed in the machine. Most incompatible machines we've encountered either have the CPU chip soldered to the motherboard or have an inaccessible CPU chip. Call for a current list of known compatible systems.

Periscope IV is available in 16MHz and 25MHz speed ratings. The 16MHz unit will work at any speed up to and including 16MHz (but not beyond!). Similarly, the 25MHz unit will work at any speed up to and including 25MHz. Do not attempt to push Periscope IV beyond its rated speed, since strange failures may occur.

Since Periscope IV gets all of its operational signals from the CPU, it is not sensitive to bus compatibility issues. However, Model IV has no access to DMA signals so it is not



---

possible to set breakpoints on DMA events or to capture them in the trace buffer. This means that memory modified by DMA cannot be detected by Periscope IV. (Periscope III gets its signals from the bus, so it does have access to DMA signals.)

**Caveats.** Setting data breakpoints can be tricky, since everything is from the perspective of the CPU. See the description of the HD command for more information.

There is a phenomenon called **breakpoint overrun** that occurs with all hardware breakpoint devices. Between the time a breakpoint is detected and the time the processor is stopped with an NMI, one or more instructions are executed. When looking in the trace buffer with the HR or HS commands, the last CPU event shown is the one that caused the breakpoint to occur. The HT and HU commands show the instructions being executed or about to be executed when the breakpoint occurred. In any event, expect that several instructions will have been executed since the breakpoint occurred. For this reason, it is pointless to set a hardware breakpoint watching for an instruction that overwrites the NMI vector. (Use Periscope's BM command for this job.) On 80386 systems, the breakpoint overrun may be up to six instructions, although it is usually two to four instructions.

**NOTE:** Don't reboot the system using Ctrl-Alt-Del when the board is armed (i.e., a GH or a GM command is in use). If a breakpoint occurs during the boot, your system may hang. Go into Periscope and use any exit other than GH or GM to disarm the board.

On an 80386 system, code prefetch cycles always start at a doubleword boundary (an address evenly divisible by four). If you set a fetch breakpoint at an address that is not on a doubleword boundary, Periscope adjusts the address to the next lower doubleword boundary.

On an 80386 CPU, the execution of an STI or a CLI instruction causes the prefetch buffer to be flushed. As a result, the trace buffer looks like it has double vision. Intel claims that this is a compatibility feature for the 8259 interrupt con-

---

troller.

80386 CPUs have varying prefetch queue lengths. In our testing, we've seen 8-, 12- and 16-byte prefetch queues. When Periscope is loaded, it displays a message showing the length of the prefetch queue if an 80386 CPU is found.

**Model IV Power Requirements.** The Model IV board requires 20 watts of power; the Model IV 80286/80386 Pod requires 3 watts of power; the Model IV 80386 Rev 1 Pod requires 3 watts of power.

## MODEL IV HARDWARE BREAKPOINT EXAMPLES

- To break in BIOS, use `HM * F000:xxxx L1 X;GH`, where `xxxx` is the desired IP in ROM. This will trap on a prefetch of the specified location, usually stopping when the instruction is about to be executed. On an 80386 system, you can also use the `BD` command to take advantage of the 80386 debug registers.
- To trap set cursor calls, use `HM 0:10*4 L4 R;BR AH EQ 1;GM`. This traps access to the video interrupt (`INT 10H`) and then tests register `AH`. If the register is equal to one, Periscope's screen is displayed, otherwise full-speed execution resumes.
- To trap writes to interrupt vectors by your program, use `HM 0:0 0:3FF W;BR CS EQ CS;GM`. If you use IBM's `VDISK` on an AT-class machine, avoid breakpoints on memory from `0:380` to `0:3FF` since this range is used as the stack while the machine returns from protected mode.
- If you suspect your program is underflowing its stack, use `HM SS:0 L3 W;GH` to trap writes to the bottom of the stack, assuming that the lower bound of the stack is at `SS:0`.
- To debug across a short boot, arm the Periscope IV board with `GH` and then restore all BIOS vectors (8, 9, 10, 15, 16, 17, 1C) to ROM and issue an `INT 19H`. Assuming `NMI` is left pointing to Periscope and Periscope's code is not corrupted by the boot, the breakpoint will re-activate Periscope. If you're loading Periscope's code into normal RAM, be sure to use the

---

/L installation option to put the code in a location that won't be corrupted by the boot. See the /Q installation option in Chapter 8.

- To monitor interrupt vector reads, enter `HM * 0:0 3FF R` and then `GH;HS`.
- To set a hardware breakpoint on 80386 shutdown, use `HM 0:0 L1 H`. To set a breakpoint on a halt, use `HM 0:2 L1 H`.
- To trap an overrun error on COM1, you can watch for the condition where port 3FDH is read and bit 1 is on. The Periscope commands to do this are: `HP 3FD I; HB MB xxxx xx1x; GH`. This sets a breakpoint on a read of port 3FD and a bit breakpoint on the mid byte when bit 1 is on. On a 286 system, modify the MB to HB to indicate the high byte.
- For more Model IV tips, see the file NOTES.TXT.

## MODEL IV IN PASSIVE REMOTE MODE

If you're using Periscope IV, you can run in passive remote mode without using the additional software and hardware required for active remote mode debugging. In the passive mode, you can debug any environment running in the target system, but you do not have full debugging capability. You can set hardware breakpoints from the host system on events in the target system, but when a hardware breakpoint occurs, the target system does not stop. You can examine the information captured in the real-time trace buffer, but you cannot access any other memory, I/O, or register information in the target system.

**NOTE:** Full Periscope/Remote debugging capability, referred to as active remote mode, is available separately as an add-on feature. It requires software and hardware not included in the standard Model IV package, and a null-modem cable that you supply. Please call The Periscope Company for details.

To run Periscope IV in passive remote mode:

- Install the Model IV board in the host system and the flex cable and/or pod in the target system.



- 
- You do not need a null-modem cable or NMI clip as with active remote mode, but you may need a longer-than-standard ribbon cable to connect the main board in the host system to the pod in the target system. The standard ribbon cable is 16 inches long, long enough to reach from host to target in some situations. The optional 48-inch shielded cable gives you three times the standard distance.
  - If you have a Plus board, install it in the host system. Periscope in passive remote mode runs out of the board's memory, if it is present.
  - Power up the target system before you install Periscope (PS.COM) on the host system.
  - When you install Periscope on the host, specify either the /286 or /386 option. Do not specify the /2:xy option. In passive remote mode Periscope runs in the host with no stub program running in the target to feed it information about the target. You must therefore specify the target CPU type when you install Periscope.
  - To keep DOS available on the host system when setting a hardware breakpoint, use the program WAITING.COM. ♦





# Keyboard Usage

- **Command Editor**
- **Keyboard Assignments**

**I**n this chapter you'll find information on Periscope's use of the keyboard, and how you can assign commands to various keys and key combinations. Periscope pre-assigns functions to some keys and key combinations, in particular the Alt-x key combinations and function keys. However, beginning with Version 5, you can assign most of the function keys (including the Alt-Fx, Ctrl-Fx, and Shift-Fx key combinations) to suit your needs. You can also configure the function keys to emulate the Turbo Debugger or CodeView function keys (see Chapter 4).

---

## COMMAND EDITOR

Periscope has a CED-like command editor. Previous commands are kept in a circular buffer that is 512 bytes in size. All lines above the minimum length set by CONFIG are saved, except when Alt-C or Alt-D is used to repeat a previous line. The following editing keys are supported:

**Backspace** — Delete the character to the left of the cursor and back up one character.

**Ctrl-End** — Erase the remainder of the command line, starting at the current cursor position.

**Ctrl-Left** — Move to the start of the previous word in the command line.

**Ctrl-PgDn** — Remove the current command line from the circular edit buffer.

**Ctrl-PgUp** — Clear the entire circular edit buffer.

**Ctrl-Right** — Move to the start of the next word in the command line.

**Del** — Delete a character from the command line.

**Down Arrow** — Display the next command line from the circular buffer.

**End** — Move the cursor to the end of the command line.

**Esc** — Erase the current command line.

**Home** — Move the cursor to the start of the command line.

**Ins** — Toggle insert mode on and off.

**Left Arrow** — Move the cursor one position to the left.

**Right Arrow** — Move the cursor one position to the right.

**Tab** — Space.

**Up Arrow** — Display the previous command line from the circular buffer.

## KEYBOARD ASSIGNMENTS

The use of function keys has been significantly changed.

Function keys F1 through F10 can now be assigned by the user. To make Periscope's key usage like that of Borland's Turbo debugger or Microsoft's CodeView, see the chapter on configuring Periscope. The default function keys are as follows:

---

KEY	ALT KEY	DESCRIPTION
F1	Alt-T	Code timing toggle
F2	Alt-N	Screen swap toggle
F3	Alt-C	Copy previous line
F4	Alt-D	Copy previous line plus return
F5	Alt-A	Display alias definitions
F6	Alt-G	Set pause mode
F7	Alt-E	Display record definitions
F8	Alt-I	Display symbols
F9	Alt-H	Call trace
F10	Alt-O	Display user screen

The ability to dynamically assign function keys to Ctrl-F1 through Ctrl-F10 using Alt-F1 through Alt-F10 has been dropped. You can now assign keys using aliases. For example, the alias name for Alt-F1 is A1. The following keys are supported:

**Alt-F1 through Alt-F10:** A1 - A9, and A0.

**Ctrl-F1 through Ctrl-F8:** C1 - C8.

(Ctrl-F9 and Ctrl-F10 are reserved by Periscope.)

**Shift-F1 through Shift-F10:** S1 - S9, and S0.

**F1 through F10:** F1 - F9, and F0.

To set Ctrl-F1 to turn off Periscope's windows, you could enter `ea c1 /w;` or have the line `\c1=/w;` in a .DEF file. If F0 through F9 are defined via aliases, the alias definition will override the default function key assignment. See the description of RS in Chapter 10 for more information on aliases.

Periscope's key usage follows.

**Alt-3 — Toggle the vertical 80386 register display on and off.**

At least one window must be used for this key sequence to have any effect.

**Alt-A — Display the alias definitions.**

If the cursor is at the beginning of the command line, all aliases are displayed. You can display alias names that start with a character sequence by entering the desired characters and then pressing Alt-A. For example, to display all alias names starting with the letter C, enter C at the start of



---

a command line and press Alt-A. Be sure not to enter any spaces before or after the search name.

**Alt-B — Toggle printer double spacing on and off when Ctrl-P is used.**

It has no effect when Shift-PrtSc is used.

**Alt-C — Copy the remainder of the previous command line into the current command line.**

This key sequence copies up to, but does not include the carriage return. The command line is not added to the circular edit buffer again.

**Alt-D — Copy the remainder of the previous command line into the current command line and add a carriage return at the end.**

For repetitive commands, you can use Alt-D.

**Alt-E — Display the record definitions.**

If the cursor is at the beginning of the 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 Alt-E. For example, to display all record definitions starting with PS, enter PS at the start of the command line and press Alt-E. Be sure not to enter any spaces before or after the search name.

**Alt-G — Select one of three pause modes: Pause on; Pause/clear on; and Pause off.**

The **Pause on** mode displays a message when the screen is full and waits for a key press before scrolling another screen full of information into view. **Pause/clear on** differs from the **Pause on** mode in that it clears the screen after a key is pressed and displays data from the top of the screen. This technique allows for much quicker updating of the second and subsequent screens, but loses the prior screen as soon as a key is pressed. The **Pause off** mode continually scrolls the non-windowed area of the screen. When loaded, Periscope defaults to **Pause on**. The pause mode is ignored when the /E (echo) mode or Ctrl-PrtSc is active.

**Alt-H — Toggle call tracing on and off.**

When this mode is turned on, the message **Call trace on** is displayed. When a GA or GT command is used, any CALL

---

instructions executed are displayed on Periscope's screen. Nested calls are indicated by leading spaces before the address, up to a maximum of 16 levels deep. For example, part of the call trace of the FTOC program (using GT) is:

170D:0016 E8BD02	CALL	___CHKSTK
170D:0072 E8A905	CALL	___PRINTF
170D:0630 E8BD01	CALL	___STBUF
170D:0640 E85103	CALL	___OUTPUT
170D:099A E839F9	CALL	___CHKSTK

This mode is best used on a dual-monitor system. A procedure label or source code may be shown along with the disassembled call, in which case the label or source code is correctly indented, but the associated call is not. When this mode is turned off, the message *Call trace off* is displayed.

#### **Alt-I — Display the address and name of the symbol table entries.**

If the cursor is at the beginning of the command line, all symbols are displayed. You can display symbols that start with a character sequence by entering the desired characters and then pressing Alt-I. For example, to display all symbols starting with the letter A, enter A at the start of a command line and press Alt-I. Be sure not to enter any spaces before or after the search name.

When local code or local data symbols are displayed using the Alt-I symbol search, an asterisk is shown immediately after the address. When a symbol is not loaded (using PLINK, .RTLink, or Microsoft Windows), it is shown in parentheses. When searching for a line number, the full name must be entered.

To search for all symbols in a segment, enter /<number>, where the number is the desired segment, and press Alt-I. To search for all symbols at a specific address, enter /<address>, where the <address> may be any of the legal address formats, and press Alt-I. To search for stack data variables, enter // [<name>], where the optional <name> is the name of the procedure, and press Alt-I. Stack data variables are not shown except when the // search is used.

#### **Alt-L — Toggle the display of commands generated by the menu**

---

system.

**Alt-M — Activate Periscope's menu system assuming that it has been enabled at CONFIG time.**

**Alt-N — Toggle screen swap on and off when Periscope is used on a single monitor.**

This key sequence has no effect when the /A, /AD, or /AV installation option is used. When off, this mode keeps Periscope from displaying the program's screen when a Jump or Trace command is used. If you're tracing code that doesn't modify the display, you may want to turn screen swap off to avoid the 'flash' caused by the restoration of the program's screen during each Jump or Trace command. Be sure to turn screen swap back on before executing code that will cause the program under test to output to the screen. If screen swap is off when a Go or Quit command is used, it is ignored. The screen is swapped anyway.

**Alt-O — Switch from Periscope's screen to the program's screen if only one monitor is being used.**

If the /A, /AD, or /AV installation option was used or if screen swap is off, this key sequence has no effect. To return to Periscope's screen from the program's screen, press any key.

**Alt-P — Generate a form feed to the parallel printer.**

**Alt-R — Toggle the vertical register display on and off.**

At least one window must be used for this key sequence to have any effect.

**Alt-S — Toggle the vertical windowed stack display on and off.**

At least one window must be used for this key sequence to have any effect.

**Alt-T — Toggle code timing on and off.**

When this mode is turned on, the message Code timing on is displayed. Your code's execution is timed in increments of 838 nanoseconds (the standard 55 millisecond timer-tick divided by 64K). On return to Periscope, the timed value is displayed as a decimal number.



---

This method is very accurate, except for very short duration events of less than 20 to 30 ticks. Due to the overhead imposed by Periscope's entry and exit code, extremely short duration events are not timed correctly. If the time displayed is less than 20 to 30 ticks or is displayed as N/A, run the stand-alone timer test program (available on request from Tech Support) or exercise the code multiple times and take an average.

The maximum event length that can be timed is 655,359,999 times 838 nanoseconds, or approximately nine minutes. The I/O ports used by this technique are 40H and 43H. To turn code timing off, press Alt-T again. The message `Code timing off` is then displayed. Note that your code is run at full speed when the code timing is on. Use of the GA or GT commands turns the code timing off. Code timing can conflict with the /K:1 installation option, since this option turns the system timer off while you're in Periscope.

**Alt-W — Toggle the current window from the data window to the watch window to the disassembly window et cetera.**

The current window is indicated by an up arrow in the separator line following the window and by a double separator line.

**Ctrl-B — Set a code breakpoint on the instruction at the top of the disassembly window.**

If a breakpoint is already set, clear it. When using overlays, be sure to set breakpoints at a symbol or source line.

**Ctrl-Break — Cancel the current Periscope command.**

**Ctrl-G — Go to the address shown at the top of the disassembly window.**

**Ctrl-PrtSc or Ctrl-P — Toggle printer echo of screen output on and off.**

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-R — Set the current instruction (CS:IP) to the instruction at**

---

**the top of the disassembly window.**

**Ctrl-S — Suspend output until another key is pressed.**

**PadMinus (gray minus key on numeric pad) — Move backward one line in the current window.**

To enter a minus sign, use the other minus key.

**PadPlus (gray plus key on numeric pad) — Move forward one line in the current window.**

To enter a plus sign, use the other plus key.

**PgDn — Page forward through the current window.**

**PgUp — Page backward through the current window.**

**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 Enter commands, you could enter `G NEWPAGE;EW PAGENO 0` to go to the procedure NEWPAGE and modify memory starting at PAGENO. After the line has been entered once, you can use Alt-D to repeat it.

**Shift-PrtSc — Prints the entire screen to the parallel printer.**

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. ♦

# Periscope's Menus

13

## ■ Menu Overview

**T**his chapter introduces the optional menus available in Periscope beginning with Version 5.

---

## MENU OVERVIEW

Periscope's menu system is an optional method for the new or occasional user to quickly and easily find and execute the appropriate Periscope command. It can also be used by seasoned users to find infrequently-used commands.

The menu system is enabled or disabled when CONFIG is run. If you don't enable the menu system, you won't be able to call it up from within Periscope. If you do enable it, you'll be able to call it up any time you need it, but it will use one line at the top of the display and load the menu system code and the on-line help and interrupt comments, which consumes approximately 60 KB of memory.

If you're using a Model I board, this has no impact on DOS memory, but if you're using a Model II or Model II-X, the larger size may affect your ability to debug your programs. You can reduce the size required by the menu system by renaming the file PSINT.DAT to some other name so that the interrupt comments won't be loaded. We do recommend that you leave the on-line help intact, but if you want to eliminate it, you can rename the file PSHELP.TXT to some other name.

If the menu system is enabled but not active, the top line of Periscope's display starts with the prompt `Alt-M: Menu` to remind you to use Alt-M to activate the menu system. The rest of this line is usually used to display a description of the function key assignments that are read from the `xxKEYS.PSD` file, where `xx` is PS, CV, or TD for Periscope, CodeView, or Turbo Debugger key usage respectively. See Chapter 12 for more information.

If the menu system is enabled, you can activate it by pressing Alt-M at the Periscope prompt. Then use the left and right arrow keys to select the desired top-level menu or use Alt-x, where x is the first character of the menu name. It is not possible to go directly to a specific top-level menu since Periscope is already using some of the Alt-key combinations for other functions.

Within a menu, you may use the up, down, home, and end

---

keys or the character corresponding to the first letter of the item name to select the desired item. (If more than one item starts with the same character, press the first letter as many times as needed to get to the desired item.)

There are three basic types of items available from the top-level menus:

**Commands that require additional information.** For these items, you'll be prompted for the additional information required. All of the usual editing keys (except escape) are available. After execution of the desired command, the previous top-level menu is displayed.

**Commands that require no additional information.** For these items, the command is immediately executed and the last top-level menu is displayed. Some items are 'toggles' and have an up or down arrow in front of the item description, indicating that they are turned on or off respectively.

**Sub-menus.** For these items, the sub-menu is displayed. Choose the appropriate entry and press the enter key. Sub-menu items are indicated by brackets instead of parentheses around the Periscope command at the end of the line. There are only two levels of menus used by Periscope — the top-level menus and one level of sub-menus.

Throughout the menu system, you can use the escape key to exit the current level. If you are at the top level menu display, the escape key returns to the Periscope prompt. If you are at a lower level in the menu system, the escape key returns you to the top level menu.

Periscope displays the command generated by the menu system, so that you can see the commands generated and learn as you use Periscope.

When Model III or IV is used, a hardware menu is added on the right-hand side of the top line. Also, when the hardware trace buffer display is active, a special hardware sub-menu is displayed.

The vast majority of Periscope commands are available

---

from the menu system. For information on commands that are not available from the menu system, please see Chapter 15.

While in the menu system, you can get help on a command by pressing F1 or ? while the desired item is selected. No help is available for sub-menus (where the command is in brackets instead of parentheses), but help is available for the items within the sub-menus. ♦

# Command Overview

14

- **Command Summary**
- **Breakpoint Command Summary**

**T**his chapter provides summaries of all commands and of the breakpoint commands. Its purpose is to give you the big picture so you won't get lost in the details of the many commands available. Chapter 15 provides syntax and descriptions for all commands; Appendix C does the same for the command parameters. For new and casual users, or users who prefer using menus, Chapter 13 introduces the optional menu system available starting with Version 5 of Periscope.

---

## COMMAND SUMMARY

Each of Periscope's commands are described on the following pages. This summary is an introduction to the various types of commands available, except for the breakpoint commands, which are covered in the next section. The commands fall into twelve groups, as follows:

**Controls:** The clear screen commands (**K** and **KI**) are used to clear Periscope's screen. The Color command (**/C**) is used to set Periscope's screen color. The Window command (**/W**) is used to set Periscope's windows. The Data window select command (**/D**) is used to select the active data window when multiple data windows are used. The Quiet command (**/Q**) is used to temporarily suppress Periscope's screen output.

**Disassembly:** The Unassemble commands (**UA**, **UB**, and **US**) disassemble memory in Assembly-only, Both-source-and-assembly, and Source-only modes respectively. You can use 16-bit (16) or 32-bit (32) disassembly as needed.

**Disk I/O:** These commands include Load Absolute (**LA**) and Load File (**LF**), the corresponding Write commands (**WA** and **WF**), and the Name command (**N**).

**Display:** The Display commands (**Dx**) display memory in various formats, including ASCII, byte, double word, effective address, integer, long real, signed integer, record, short real, word, long integer, signed long integer, and ASCIIz. The Watch commands (**w**) are used to display memory in any of the above formats in a watch window.

**Execution:** These commands include the Trace and Trace Line commands (**T** and **TL**), the Jump and Jump Line commands (**J** and **JL**), and the Go commands (**Gx**) that execute your program under various conditions. To trace all but interrupts, use the (**TI**) command. The Breakpoint commands (**Bx** and **Hx**) are used to set code, monitor and hardware breakpoints. See the discussion of Periscope's breakpoint commands in the next section of this chapter.

**Hardware:** The Hardware commands (**Hx**) are used with



---

**Periscope IV** to set hardware breakpoints and controls and to display the hardware trace buffer. See the separate addendum for Model III hardware commands.

**I/O:** The Input (**I** and **IW**) and Output (**O** and **OW**) commands are used to read and write I/O ports, respectively.

**Modify:** Various commands are used to modify memory, including the Enter commands (**E**, **EB**, **ED**, and **EW**), the Fill command (**F**), and the Move command (**M**).

**Search:** The Search commands search memory in a variety of ways, including two data searches (**S** and **SD**), an address search (**SA**), a disassembly search (**SU**), and two stack searches, one for calls (**SC**) and one for returns (**SR**).

**Status:** The status commands include the Register command (**Rx**) which is used to display, change, save, compare, and restore the machine registers and flags. The software trace buffer is viewed using the Traceback commands (**TB**, **TR**, and **TU**). The interrupt vectors can be saved, compared, and restored using the **IR**, **IC**, and **IS** commands. To save the state of a debug session, use the **WB** command. To later restore the state, use the **LB** command.

**Symbols:** The symbol table is loaded and written using the Load Symbols (**LS**) and Write Symbols (**WS**) commands. The Load Definitions (**LD**) and Write Definitions (**WD**) commands load and write the record definition table. The Enter Symbol command (**ES**) is used to add symbols on the fly. The Near command (**/N**) is used to search for the nearest symbols in memory to a specified address. The Remove symbol command (**/R**) is used to delete a symbol from the symbol table. The Segment change command (**/S**) is used to modify the segment for a group of symbols.

**Miscellaneous:** The miscellaneous commands include an inline Assembler (**A** and **AU**), a Compare memory command (**C**), an Enter Alias command (**EA**), Hex arithmetic (**H**), Quit (**Qx**), View file (**V** and **VS**), and translate addresses and numbers (**XA**, **XD**, and **XH**).

Also available are **option (/)** commands, including the

---

Echo command (/E) which echoes Periscope's screen output to a disk file. The Message command (/M) sets the message level when Microsoft Windows is used. The User exit command (/U) is used to execute user-written code from within Periscope. Finally, the eXit command (/X) is used to exit to DOS. See also the Symbols commands above.

The key capture command (/K) captures keyboard input to a file. The /1 and /2 commands allow you to switch to the first and second symbol tables, respectively. The DOS access command (/A) turns DOS access on and off. To copy Periscope's screen to the other display, use the dupe (/") command. To toggle the display of line-number symbols and source-level debugging status, use the /L command.

## BREAKPOINT COMMAND SUMMARY

There are four categories of breakpoints: Code breakpoints, Monitor breakpoints, Hardware breakpoints, and Debug register breakpoints. **Code breakpoints** are set on specific addresses in your program. When the instructions at the specified addresses are executed, a breakpoint is taken. **Monitor breakpoints** can be set on a wide class of dynamically-evaluated conditions. **Hardware breakpoints** are set with the Periscope III and Periscope IV hardware only. They are set on the real-time bus (Model III) or CPU (Model IV) events that occur during the execution of your program. **Debug register breakpoints** are available beginning on 80386 machines. These breakpoints are set on reads, writes, and executes of memory and run in real-time.

**Code breakpoints** are set via the BC (Breakpoint on Code) command. These breakpoints are 'sticky' since they are remembered until explicitly cleared with the BC or BA commands. Temporary code breakpoints are set when an address is entered with the G (Go) command. They are not remembered after the execution of the G command that set them. All of the Go commands activate Code breakpoints. Unless used with the GA, GM, or GT commands, these breakpoints run the system at full speed.

**Monitor breakpoints** are set via these commands: BB (Byte), BF (Flag), BI (Interrupt), BL (Line), BM (Memory), BP (Port), BR (Register), BU (User), BW (Word), and BX

(eXit). They are all 'sticky', or remembered until explicitly cleared. They are activated by the GA, GM, and GT commands only. They cause the system to run much slower than full speed.

**Hardware breakpoints** are set via the HB (Hardware Bit), HC (Hardware Controls), HD (Hardware Data), HM (Hardware Memory), and HP (Hardware Port) commands. All Hardware breakpoints are 'sticky', or remembered until explicitly cleared. These breakpoints are activated by the GH and GM commands only. They run at full speed except when GM is used.

**Debug register breakpoints** are set via the BD (Breakpoint on 80386 Debug Registers) command. If you're using an 80386 machine and want to use the debug registers, be sure to indicate this when you configure Periscope. See the description of the BD command for more information on setting these breakpoints. They are activated by all of the Go commands, and run at full speed unless the GA, GM, or GT commands are used.

GO COMMAND	BREAKPOINT TYPES	SPEED
G	C,D	Full speed
G+	C,D	Full speed
G=	C,D	Full speed
GA	C,D,M	Very slow
GH	C,D,H	Full speed
GM	C,D,M,H	May be slow
GR	C,D	Full speed
GT	C,D,M	Slow

BREAKPOINT TYPES: C=Code; D=Debug register; M=Monitor; H=Hardware

*Figure 14-1. Execution Speeds of Various Breakpoints*

Figure 14-1 summarizes program execution speeds when the various types of breakpoints are set and the various Go commands are used.

---

Breakpoint command terms that may be confusing at first are: set, clear, enable, and disable. Here's what they mean. When you **set** any 'sticky' breakpoint, it will be in effect each time you use the appropriate Go command. You can keep it from being in effect by either disabling or clearing it. If you **clear** it, Periscope no longer knows about it. If you **disable** it, Periscope knows it's there, but does not use it until you **enable** it. Disabled breakpoints are displayed with a leading minus sign.

To clear all breakpoints, enter **BA \*** (for software breakpoints) or **HA \*** (for hardware breakpoints). To clear an entire group of breakpoints, enter **Bx \*** or **Hx \***, where **x** indicates the group you want to clear, such as Byte, Memory, Port, Word, etc. To clear an individual breakpoint, re-enter the breakpoint (except for the **HB** and **HC** breakpoints). Periscope's program loader, **RUN**, always disables all hardware and software breakpoints, except for the **HC** breakpoints. A useful habit to develop is to display all breakpoints with **BA** and **HA** before executing the Go command, so that you know for sure what breakpoints are set and enabled. ♦

# Command Reference

15

## ■ Commands

**T**his chapter lists all Periscope commands. You can use most of these commands regardless of the model of Periscope you're using. Those that require Model IV are so designated. You will most likely find it helpful to read the Keyboard Usage, Periscope's Menus, and Command Overview chapters before you delve into the command details in this chapter.

---

## COMMANDS

### Command: help (?)

**Syntax:** ? [`<command>`]

**Description:** This command displays Periscope's commands if the on-line help file has been loaded. If the help file is not available, a command summary is displayed.

**Examples:**

? displays a command summary.

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

### Command: Assemble to memory (A)

**Syntax:** A [`<address>`]

**Description:** This command assembles instructions to memory.

To use the in-line assembler, enter A [`address`] when Periscope's prompt is displayed. The specified `<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 real-mode opcodes up through the 80286 and 80287. The protected-mode opcodes of the 80286 and 80386, and later opcodes, are not supported. If the LOCK 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 may be preceded by 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. Some 8087 and 80287 instructions require a width indicator of `D`, `Q`, or `T` for Double word, Quad word, and Ten byte, respectively.

Periscope also supports the `DB` pseudo-op, allowing you to enter hex or ASCII characters into memory.

### **Example:**

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

## **Command: Assemble then Unassemble (AU)**

**Syntax:** `AU [<address>]`

**Description:** This command is the same as the Assemble command described previously, except that it disassembles

---

an instruction immediately after assembling 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: software Breakpoints All (BA)**

**Syntax:** BA [?] [\*] [+] [-] [A]

**Description:** This command displays (?), clears (\*), enables (+), disables (-), and/or toggles (A) the monitor breakpoint AND mode for the currently-set software breakpoints. If you enter BA only, display is assumed, and all currently-set breakpoints are displayed. Disabled breakpoints are displayed with a leading minus sign.

The AND mode ANDs the results of monitor breakpoints in different classes. Multiple breakpoints within a class (i.e. multiple register breakpoints) are still ORed together, but the result of one class is ANDed with the results of other classes.

Each class of currently-used monitor breakpoints must indicate a 'hit' for a breakpoint to be generated when the AND mode is on. For example, if the BB, BI, and BM breakpoints are all set and enabled, all must be true for a breakpoint to be generated when AND mode is on. If multiple breakpoints are set within a class, such as two register breakpoints, only one true condition within the class is needed to make the entire class true.

**NOTE:** Since RUN disables all breakpoints, use this command to re-enable previously-set breakpoints.



---

### Examples:

Assume that a Byte breakpoint has been set for the symbol `LINECOUNT` equal to `38H` and that a Register breakpoint has been set for `CX` less than 5. No other breakpoints have been set.

`BA` or `BA ?` displays both breakpoints:  
`BB LINECOUNT EQ 38` and `BR CX LT 0005`.

`BA *` clears both breakpoints.

`BA +` enables both breakpoints. (Line and eXit breakpoints are enabled only if they have been previously turned on with `BL +` or `BX +` and then disabled.)

`BA -` disables both breakpoints. (Line and eXit breakpoints are disabled only if they have been previously turned on with `BL +` or `BX +`.)

Assume you want to watch for a `RETurn` where `SP` is greater than or equal to the current value:

`BA * A; BX +; BR SP GE SP` clears all breakpoints, sets `AND` mode on, sets the exit breakpoint on, and sets a breakpoint when `SP` is greater than or equal to its current value.

## Command: Breakpoint on Byte (BB)

**Syntax:** `BB [<address> <test> <byte>] [?] [*] [+] [-] [...]`

**Description:** This command is used to set a monitor 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 seg-

---

ment is used. If any of the tests pass, a breakpoint is taken. 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. Re-entering a previously set breakpoint clears the breakpoint and displays the message `Breakpoint cleared`. Be careful to display all breakpoints before using the Go command to make sure the breakpoints you've got are the ones you want.

### Examples:

`BB LINECOUNT EQ 38` sets a Byte breakpoint for the memory location corresponding to `LINECOUNT`.

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

`BB` or `BB ?` displays all Byte breakpoints.

## Command: Breakpoint on Code (BC)

**Syntax:** `BC [<address>] [?] [*] [+] [-]`  
`[!<number>] [...]`

**Description:** This command is used to set a code breakpoint when an instruction is executed. A software pass count can be specified by entering a number from 1 to FFFFH after the exclamation mark.

It performs the same function as addresses entered after the Go command, except that these breakpoints are remembered. If a segment is not specified in the address, CS is presumed. Any form of the Go command may be used to enable Code breakpoints, but only the `GM` command ac-

---

tivates the software pass counter. 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. Re-entering a previously set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the Go command to make sure the breakpoints you've got are the ones you want.

When a code breakpoint is set, the instruction is highlighted in the disassembly window unless it is the current line, in which case an arrow at the start of the line is used.

### Examples:

**BC PRINTLINE** sets a Code breakpoint for the memory location corresponding to PRINTLINE.

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

**BC {0:21\*4** sets a Code breakpoint at the entry point for Interrupt 21H.

**BC** or **BC ?** displays all Code breakpoints.

## Command: Breakpoint on 80386 Debug Registers (BD)

**Syntax:** **BD** [**<address>**] [**L<byte> R|W|X**] [**?**] [**\***] [**+**] [**-**] [**...**]

**Description:** This command lets you take advantage of the 80386 debug registers to set up to four limited real-time hardware breakpoints on instruction execution, memory writes, or memory access (read or write). In the syntax, **<byte>** is 1, 2, or 4, representing a byte, word, or double



---

word respectively. The R, W, and X indicate read/write, write, and execute respectively. The segment portion of the <address> defaults to DS. Be sure to override it as needed.

Be sure to set execution breakpoints on the first byte of the instruction, including any prefixes. For read/write breakpoints, proper alignment is necessary to avoid missed breakpoints. If the length is 2, the address should be on a word boundary. Similarly, if the length is 4, the address should be on a double word boundary. This breakpoint is active whenever code breakpoints are active.

To use this command, you must have an 80386 or later CPU and you must have indicated that you want to use the debug registers when you configured Periscope.

If you're using a Compaq 80386 Deskpro with CEMM, you may encounter problems with the debug register support. If you get an exception interrupt with CS:IP inside Periscope, you've run into the problem. The debug register code requires use of instructions that are not properly handled by CEMM. If you run into this problem, you'll need to disable the debug register support using CONFIG or, better yet, get another memory manager for the 80386. One highly-regarded memory manager is 386MAX by Qualitas, Inc., of Bethesda, Maryland.

Do not reboot the system while a BD breakpoint is in effect, since it can hang the system. If the breakpoint is hit during the boot process, the system tries to access Periscope, which is no longer resident. In general, you should try to disarm BD breakpoints before your program terminates.

### **Examples:**

**BD F000:FEED** sets a breakpoint on instruction execution in ROM.

**BD 0:46C L2 W** sets a breakpoint on a write to the system clock.

---

## Command: Breakpoint on Flag (BF)

**Syntax:** BF [<flag>] [?] [\*] [+] [-]

**Description:** This command is used to set a monitor breakpoint on a single <flag> register value. For example, to check for unsuccessful DOS calls where the carry flag is set, set a code breakpoint at the return point, enter **BF CY** to set a flag breakpoint, and enter **GM** to begin execution. Only one flag breakpoint can be set at any time. 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. Be careful to display all breakpoints before using the Go command to make sure the breakpoints you've got are the ones you want.

### Examples:

**BF OV** sets a breakpoint when the overflow flag is on.

**BF -** disables the flag breakpoint.

## Command: Breakpoint on Interrupt (BI)

**Syntax:** BI [<byte>] [?] [\*] [+] [-] [#]  
[...]

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

This breakpoint is used to get to the next occurrence of any software interrupts from 0 to FFH. The interrupt number is entered as <byte>. Multiple interrupt numbers may be entered on a single line. To set breakpoints on all interrupts, enter **BI #**. The breakpoint clear (\*), display (?), enable (+), and disable (-) functions may also be present on the line. If the interrupt of interest is in RAM, don't use this breakpoint. Instead use a Go command to get to the in-

---

interrupt at full speed. For example, to get to Int 21H, enter  
**G {0:21\*4}**.

After being set, these breakpoints are remembered until they are cleared. Re-entering a previously set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the Go command to make sure the breakpoints you've got are the ones you want.

### **Examples:**

**BI \* 13** clears all Interrupt breakpoints and then sets a breakpoint on Interrupt 13H.

**BI #** sets all Interrupt breakpoints.

**BI or BI ?** displays all Interrupt breakpoints.

## **Command: Breakpoint on Line (BL)**

**Syntax:** **BL** [?] [\*] [+] [-]

**Description:** This command is used to set a monitor breakpoint when a source code line is reached.

This breakpoint is used to get to the next instruction that corresponds to a source line of a 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 (but slow) method of getting back to the source program. It requires source line numbers to be in the symbol table. After setting the Line breakpoint, use **GA** or **GT** to execute to the next source line. Since this command can be very slow, use a **G** command to a known execution address if possible.

**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.

**NOTE:** You may also use the `SR` and `SC` commands to analyze the stack to determine the next source line.

**Example:**

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

## Command: Breakpoint on Memory (BM)

**Syntax:** `BM [<address> <address> R and/or W and/or X] [?] [*] [+] [-] [...]`

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

The two addresses may have different segments, but the second `<address>` must not be lower in memory than the first `<address>`. A range may be used instead of the two addresses. 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 read (`R`) or write (`W`) breakpoints will occur only if a read or write starts in the specified range. The execute (`X`) breakpoint will occur only if `CS:IP` is in the specified range. If any of the tests pass, a breakpoint is taken. This breakpoint stops execution of the program on the instruction that will read, write, or execute 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.

This breakpoint will not detect a change caused by code that is not traced. It will never see changes made by a hardware interrupt. If you're using the `GT` command, be

---

sure that the appropriate interrupts are being traced.

After being set, these breakpoints are remembered until they are cleared. Re-entering a previously set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the **Go** command to make sure the breakpoints you've got are the ones you want.

### **Examples:**

**BM DATASTART DATAEND W** sets a Memory breakpoint for writes 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.

**BM or BM ?** displays all Memory breakpoints.

## **Command: Breakpoint on Port (BP)**

**Syntax:** **BP** [<port> [<port>] **I** and/or **O**] [?] [\*] [+] [-] [...]

**Description:** This command is used to set a monitor 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. 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. Re-entering a previously set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the Go command to make sure the breakpoints you've got are the ones you want.

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

**BP or BP ?** displays all Port breakpoints.

## Command: Breakpoint on Register (BR)

**Syntax:** BR [<register> <test> <number>] [?] [\*] [+] [-] [...]

**Description:** This command is used to set a monitor 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. 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 8-bit or 16-bit registers except FS and GS 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. Re-entering a previously set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the Go command to make sure the breakpoints you've got are the ones you want.

### 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. DS is used for its current value only.

**BR** or **BR ?** displays all Register breakpoints.

## Command: Breakpoint on User test (BU)

**Syntax:** BU [**<number>**] [**?**] [**\***] [**+**] [**-**] [**...**]

**Description:** This command is used to enable a user-written monitor 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 10 must be installed before Periscope is run. Also, the /I installation option must be used when Periscope is installed. When a User breakpoint command is entered, Periscope displays an error if the signature of the user exit code cannot be found. On return from the user routine, register AL must be set to 1 if a breakpoint is to be taken. Any other value causes no breakpoint to be taken.

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. Re-entering a previously set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the Go command to make sure the breakpoints you've got are the ones you want.

### Examples:

Assuming that a user-written interrupt handler has been installed using INT 60H and that Periscope 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.

BU or BU ? displays all User breakpoints.

## Command: Breakpoint on Word (BW)

**Syntax:** BW [<address> <test> <number>] [?] [\*] [+] [-] [...]

**Description:** This command is used to set a monitor 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. 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. Re-entering a previously set breakpoint clears the breakpoint and displays the message `Breakpoint cleared`. Be careful to display all breakpoints before using the `Go` command to make sure the breakpoints you've got are the ones you want.

**Examples:**

`BW CHARCOUNT EQ 1234` sets a Word breakpoint for the memory location corresponding to `CHARCOUNT`.

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

`BW` or `BW ?` displays all Word breakpoints.

## Command: Breakpoint on eXit (BX)

**Syntax:** `BX [?] [*] [-] [+]`

**Description:** This command is used to set a monitor breakpoint on return from a subroutine or interrupt handler.

With this breakpoint set, execution continues until a `RET`, `RETF`, or `IRET` instruction is found. It is a convenient method of executing until the program is about to transfer control to another procedure.

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

**Examples:**

`BX +` turns the eXit breakpoint on so that a subsequent `GA` or `GT` command will stop when a `RET`, `RETF`, or `IRET` instruction is reached.

---

**BX** or **BX ?** displays the status of the eXit breakpoint.

## Command: Compare (C)

**Syntax:** **C** <range> <address>

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

If any differences are found, the address and value of the first byte and the value and address of the second byte is displayed. Nothing is displayed for bytes that match. Since this command accepts two addresses as input, the two blocks of memory may be in different segments. If no segment is input, the current data segment is used. The length part of the <range> 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 com-

---

pared 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 (D)

**Syntax:** D [<range>]

**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 following three paragraphs.

The syntax for all of the Display commands except DE and DR is very flexible. If you enter Dx, where x is the format, 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. If a data window is used, the size of the window overrides the <length> parameter.

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 of the skipped lines.

If a data window is used and the window is active, the PgUp, PgDn, PadMinus, and PadPlus keys can be used to

---

move forward and backward through memory.

**Examples:**

**D** displays memory starting where the last Display command left off.

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

**D LINECOUNT L 1** displays memory starting at the symbol LINECOUNT for a length of 1 using the current format.

## **Command: Display using ASCII format (DA)**

**Syntax:** **DA** [**<range>**]

**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.

If a data window is used with this format, the PgUp and PadMinus keys do not keep the display aligned, since the data is variable length.

**Examples:**

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

**DA FILENAME L20** displays memory starting at the sym-



---

bol 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 (DB)**

**Syntax:** DB [<range>]

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

Each line of the display shows the starting segment and offset, up to 16 bytes, and their ASCII representation. A dash is displayed between the eighth and ninth bytes for readability.

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.

### **Examples:**

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

DB LINECOUNT L 1 displays the byte at the symbol LINECOUNT.

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

## **Command: Display using Double word format (DD)**

**Syntax:** DD [<range>]



---

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

This format is useful for examining data that is stored as a word offset followed by a word segment. Each line of the display shows the starting segment and offset and up to four pairs of segments and offsets.

This command shows your program's interrupt vectors for interrupts 8, 9, 10H, 15H, 16H, 17H, 1BH, 1CH, 23H, and 24H. An asterisk after the doubleword indicates this is a virtual value. The other display commands show the real values, which are Periscope's current values.

**Examples:**

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

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

## Command: Display Effective address (DE)

**Syntax:** DE

**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 (DI)

**Syntax:** `DI [<range>]`

**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.

### 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 Long real format (DL)

**Syntax:** DL [<range>]

**Description:** This command is used to display a block of memory in long real (quad word) format.

This format is used to examine data that is stored as an 8-byte floating point number in 8087 (IEEE) format. Each line of the display shows the starting segment and offset and up to two numbers in scientific notation.

### Examples:

DL displays memory starting where the last Display command left off.

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

DL ARRAY displays memory starting at the symbol ARRAY.

## Command: Display using Number format (DN)

**Syntax:** DN [<range>]

**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).



---

### 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 (DR)

Syntax: DR [!] <address> <symbol>

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

---

\PSP	; Program Segment Prefix
Int 20,b,2	; DOS return
Topmem,w,2	; Amount of memory in paragraphs
Res.,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
Term Addr,d,4	; CS:IP of DOS terminate address
Brk Addr,d,4	; CS:IP of Ctrl-Break exit address
Err Addr,d,4	; CS:IP of critical error exit address
Res.,b,16	; Reserved for DOS
Environ,w,2	; DOS 2.00 Environment segment
Res.,b,2e	; Reserved for DOS
FCB1,b,10	; The first FCB read from the command line
FCB2,b,14	; The second FCB read from the command line

---

*Figure 15-1. Definition of the PSP from the File PS.DEF*


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

used, the address of each field is displayed on the line before the field.

To use a record format, a record definition, or DEF file, must exist. RUN loads the record definitions from the DEF file. You can add record definitions to the DEF file using a text editor. See the sample file PS.DEF and the description of RS.COM in Chapter 10.

Enter `DR CS:0 PSP` to get a display similar to Figure 15-2.

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.



Int 20	CD 20	M
Topmem	A000	
Res.	00	.
Long Call	9A	.
DOS Func	F01D:FEF0	
Term Addr	113E:012F	
Brk Addr	113E:013C	
Err Addr	113E:04C4	
Res.	3E 11 01 01 01 00 02 04-05 FF FF FF FF FF FF FF FF ).....	
	FF FF FF FF FF FF	.....
Environ	2F8B	
Res.	E6 04 05 19 14 00 18 00-93 2F FF FF FF FF 00 00 f.U...../.....	
	* 0001 Lines Of 00 Skipped *	
	00 00 CD 21 CB 00 00 00-00 00 00 00 00 00	..M!K.....
FCB1	00 20 20 20 20 20 20 20-20 20 20 20 00 00 00 00	.....
FCB2	00 20 20 20 20 20 20 20-20 20 20 20 00 00 00 00	.....
	00 00 00 00	....

Figure 15-2. Sample Display Using the DR Command

**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 Short real format (DS)**

**Syntax:** DS [<range>]

**Description:** This command is used to display a block of memory in short real (double word) format.

This format is used to examine data that is stored as a 4-byte floating point number in 8087 (IEEE) format. Each line of the display shows the starting segment and offset and up to two numbers in scientific notation.

### **Examples:**

**DS** displays memory starting where the last Display command left off.

**DS DS:SI L 10** displays memory starting at DS:SI for a length of 10H bytes.

**DS ARRAY** displays memory starting at the symbol ARRAY.

## **Command: Display using Word format (DW)**

**Syntax:** DW [<range>]

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

---

This format is useful for examining data that is stored as words 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.

**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: Display using long integer format (DX)**

**Syntax:** DX [<range>]

**Description:** This command displays memory in long integer (four-byte) format. This format is useful for examining data that is stored as an unsigned long integer.

Each line of the display shows the starting segment and offset and up to 4 decimal numbers. The number displayed may be from zero to 4,294,967,295.

**Examples:**

DX displays memory starting where the last Display command left off.

DX ARRAY displays memory starting at the symbol ARRAY.

---

## Command: Display using long signed integer format (DY)

**Syntax:** DY [<range>]

**Description:** This command displays memory in a long signed integer (four-byte) format.

Each line of the display shows the starting segment and offset and up to four decimal numbers. The number displayed may be from -2,147,483,648 to +2,147,483,647.

### Examples:

DY displays memory starting where the last Display command left off.

DY DATAPOINTS displays memory starting at the symbol DATAPOINTS.

## Command: Display using asciiZ format (DZ)

**Syntax:** DZ [<range>]

**Description:** This command is used to display a block of memory in nul-terminated ASCII format.

This command is the same as the DA command, except that the display ends when a nul (binary zero) is found. If a data window is used, the display continues to the end of the data window. See the description of the DA command above for more information.

### Examples:

DZ displays memory starting where the last Display command left off.

DZ FILENAME displays memory starting at the symbol



---

FILENAME and continues until a nul is found or 80H bytes have been displayed.

## Command: Enter (E)

**Syntax:** E <address> [<list>]

**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. Periscope 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. 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.

### 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 Alias (EA)

**Syntax:** EA <alias> [<name>]

**Description:** This command is used to define or redefine an alias, which is a two-character shorthand notation for a one-to-64-character <name>. If this command is used without a name parameter, the <alias> is deleted.

The **X0**, **X1**, **X2**, and **X3** aliases, which cause Periscope commands to be executed when **RUN** is used, on entry to Periscope, after each Periscope command, and on exit from Periscope respectively, are reserved. The **MP** and **MX** aliases, which set the module path and module extension for source-level debugging, are also reserved.

To display the current aliases, press the Alt-A keys. You can assign other aliases as needed and then activate them as commands by using ^XX, where XX is the two-character alias name.

For more information on aliases, see RS in Chapter 10 and the <alias> command parameter in Appendix C.

### Example:

EA X2 D ES:DI defines alias **X2** as D ES:DI. This command is executed after each Periscope command, a convenient way to constantly monitor the current value of ES:DI.

---

## Command: Enter Bytes (EB)

**Syntax:** EB <address> <list>

**Description:** This command is used to modify memory a byte at a time. It is similar to the Enter command, but has no interactive mode. Each item in the <list> is a byte value. One or more bytes may be entered. The segment and offset must be specified for the <address>.

**Example:**

EB FILENAME 'A:FILE' 0 modifies memory at the symbol FILENAME to contain the nul-terminated string A:FILE.

## Command: Enter Doublewords (ED)

**Syntax:** ED <address> <address> [...]

**Description:** This command is used to modify memory a doubleword at a time. It is similar to the Enter command, but has no interactive mode. Each <address> in the command line is composed of a segment and offset or a symbol. The segment and offset must be specified for the first address, which is the destination. The second and subsequent addresses are the values to be written.

**Example:**

ED 0:0 1234:5678 modifies INT 0 (divide overflow) to point to 1234:5678.

## Command: Enter Symbol (ES)

**Syntax:** ES <address> <symbol>

---

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

A segment and offset must be specified for the <address>. The <symbol> name must be 32 characters or less and may be preceded by a period. This command adds symbols to the end of the symbol table, regardless of any duplicate names in prior public, line, or local symbols. Use the /R command to delete any conflicting symbol names.

**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: Enter Words (EW)

**Syntax:** EW <address> <number> [ ... ]

**Description:** This command is used to modify memory a word at a time. It is similar to the Enter command, but has no interactive mode. Each <number> is a word. One or more words may be entered. The segment and offset must be specified for the <address>.

**Example:**

EW ARRAY 1 2 3 writes three words starting at ARRAY.

---

## Command: Fill (F)

**Syntax:** F <range> <list>

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

A segment and offset must be specified for the address portion of the <range>. 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 (G)

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

**Description:** The Go command is used to set temporary code breakpoints, activate sticky code breakpoints and debug register breakpoints, and execute the program being debugged. (See the Breakpoint Commands Summary in the previous chapter for a general discussion of breakpoints and breakpoint terminology.) This command activates code breakpoints and debug register breakpoints only. To activate monitor breakpoints, use the GA, GM or GT com-

---

mands. To activate hardware breakpoints, use the **GH** or **GM** commands.

**NOTE:** You can use **Ctrl-G** to go to the line displayed at the top of the disassembly window.

If any **<address>** parameters 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. This method allows you to set up to 16 sticky code breakpoints. To set debug register breakpoints, use the **BD** command. This command allows you to set up to four limited real-time hardware breakpoints on instruction execution, memory writes, or memory access (read or write).

If **G** with no addresses is entered, the sticky code and debug register breakpoints, if any, are used. If there are no sticky code or debug register breakpoints, program execution continues until completion or until the break-out switch is pressed.

Code breakpoints are remembered until cleared or until Periscope is rerun. If you have code or debug register breakpoints set and want to continue program execution without using any of the breakpoints, you can disable all breakpoints using **BA -**, or you can use the **QC** command.

You cannot set code breakpoints in ROM. (Use the debug register breakpoints to do this!) Code breakpoints require that Periscope be able to exchange the original byte with **CCH** before starting the **Go** command. Since the setting of a code breakpoint in the middle of an instruction can have unpredictable results, set code breakpoints using symbol names where possible.

---

### Examples:

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

## Command: Go plus (G +)

**Syntax:** G+

**Description:** This command sets a temporary code breakpoint on the next instruction, activates any code breakpoints set with the BC command, and executes the program being debugged. G+ activates code and debug register breakpoints only. Other than setting a temporary breakpoint on the following instruction, it is identical in function to the G command described above.

### Example:

If IP is 200H and the current instruction is INT 21H:

G+ sets a temporary breakpoint at 202H, which is after the INT instruction.

---

## Command: Go equal (G=)

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

**Description:** This command sets CS:IP to the first <address>, sets any indicated temporary code breakpoints, activates any sticky code breakpoints set with the BC command, and executes the program being debugged. This command activates code and debug register breakpoints only.

Other than setting CS:IP to the first address entered, this command is identical in function to the G command described above. The equal sign must appear immediately after the letter G.

### Examples:

G=PRINTF sets CS:IP to the value indicated by the symbol PRINTF and executes the program.

G=123 sets CS:IP to CS:123 and executes the program.

## Command: Go using All (GA)

**Syntax:** GA [<address>] [...]

**Description:** This command is the same as the GT command, except that ALL instructions are traced by this command. This command activates code breakpoints, debug register breakpoints, and monitor breakpoints. The GT command single steps through instructions, except when a software interrupt is performed. Then the interrupt trace table (see the /T command) is checked. If the interrupt is not in the table, GT does not trace through the interrupt. This can cause GT to miss breakpoints. The GA command always traces ALL the way through ALL software interrupts. It is slower than GT, but more dependable.

**NOTE:** It is not possible to trace into hardware interrupts with this command. Use Periscope III or



---

IV for this purpose.

**Example:**

See the examples under the GT command.

## **Command (Model IV): Go using Hardware (GH)**

**Syntax:** GH [<address>] [...]

**Description:** This command is used to activate code breakpoints, debug register breakpoints, and hardware breakpoints (not monitor breakpoints) and execute the program being debugged. It is the same as the Go command, except that it also activates any hardware breakpoints that are enabled. The hardware breakpoints are disabled when RUN is used. Be sure to check the status of the hardware breakpoints using the HA command before starting execution.

**Examples:**

GH PRINTLINE sets a temporary code breakpoint at the address equal to the symbol PRINTLINE, invokes all code and hardware breakpoints that are set and starts execution of the program.

GH invokes all code and hardware breakpoints that are set and begins execution of the program.

## **Command: Go using Monitor (GM)**

**Syntax:** GM [<address>] [...]

**Description:** This command is used to go at full speed to a certain point and then evaluate the monitor breakpoints. If any of the monitor breakpoints indicate a hit, Periscope's

---

screen is displayed. Otherwise full speed execution resumes. This command activates code breakpoints, debug register breakpoints, and hardware breakpoints. The monitor breakpoints are evaluated only when a code, debug register, or hardware breakpoint occurs.

For all models of Periscope other than Model III and Model IV, this command acts as a combination of the **G** and **GT** commands. Consider the situation where you need to watch a buffer for an end of file marker. Using **GT**, this would usually be very time-consuming. If you enter a monitor test (such as **BR AL EQ 1A**) and then use **GM** to go to the appropriate place in your code, most of the code will be executed at full speed. Each time a code or debug register breakpoint is reached, the monitor breakpoint(s) are evaluated, rather than after every instruction.

For Periscope III and Periscope IV, this command acts like a combination of the **GH** and **GT** commands. Consider a situation where you need to find where your program is writing to low memory. Since DOS and other programs can be legitimately writing to low memory, you need to watch for writes to low memory where the offending Code Segment points to your program. To do this with Periscope III or IV, set a hardware breakpoint watching for writes to the desired range, then enter a monitor breakpoint (such as **BR CS EQ CS**), and then enter **GM**. Each time a hardware breakpoint occurs, the monitor breakpoint(s) are evaluated. If any of the monitor breakpoints indicate a hit, Periscope's screen is displayed. Otherwise full speed execution resumes. The monitor breakpoints of interest when Periscope III or IV is used are **BB**, **BF**, **BR**, **BW**, and **BU**. The Register breakpoint is particularly powerful, since register information is not available at the hardware level. For more complex events, the user breakpoint tests can be used. This command can run slowly if the hardware breakpoint occurs frequently. Try to qualify the hardware breakpoint as tightly as possible for best performance. Also, see the description of the program **SKIP21** in Chapter 10 for another method of trapping writes to low memory.

**NOTE:** Each time a hardware breakpoint occurs, a discontinuity appears in the real-time trace buffer. This happens because nothing is added to the

---

buffer from the time the hardware breakpoint occurs until just before Periscope returns control to the interrupted program.

Since the monitor breakpoint is evaluated one or more instructions after the instruction that caused the hardware breakpoint, it is possible for the hardware breakpoint and the software breakpoint to be out of sync. This can cause an occasional false or missed breakpoint. For example, if you're watching for writes to memory where the CS points to your program, code that changes CS during the breakpoint overrun interval can cause problems.

**Example:**

GM PRINTLINE sets a temporary code breakpoint at the address equal to the symbol PRINTLINE, invokes all hardware breakpoints that are set (if Periscope III or IV is used) and starts execution of the program. When a code, debug register, or hardware breakpoint occurs, the monitor breakpoints are evaluated. If any of the monitor breakpoints indicate a hit, Periscope's screen is displayed. Otherwise, full-speed execution resumes.



## **Command: Go to Return address on stack (GR)**

**Syntax:** GR

**Description:** This command analyzes the stack for the first return address and sets a temporary code breakpoint at that address.

This command uses the same logic as the SR command. If no return address is found, an error is displayed. This command has no arguments. Any addresses entered after GR are ignored.

**Example:**

GR examines the stack for a return address and then goes to

---

that address.

## Command: Go using Trace (GT)

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

**Description:** This command is the similar to the normal Go command, except that it enters a single-step mode and evaluates the monitor breakpoints after each instruction. This command activates code breakpoints, debug register breakpoints, and monitor breakpoints.

This command 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 (unless you're using Periscope III or IV). Since this command is slow, try to use the normal Go command to get as close to the problem as possible, then use GT.

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

To make sure you've got the correct breakpoints, get in the habit of checking the breakpoint settings before using this command. Enter BA to display the current breakpoints before entering GT.

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

When a software interrupt is encountered, it is executed step-by-step if and only if the interrupt number is set in the trace table (see the description of the /T command). If the

---

breakpoint you're trying to find is in an interrupt or is caused by an interrupt, you should use the **GA** command, since it traces all software interrupts.

**NOTE:** It is not possible to trace into hardware interrupts with this command. Use Periscope III or IV for this purpose.

If you're using the **GT** command and the program being debugged starts running at full speed, an ill-behaved interrupt has probably modified the trap (single-step) flag. To find the problem interrupt, press the break-out switch and then use **TB** to see the last code traced by Periscope. Note the last entry in the software trace buffer. The interrupt shown is the one that caused Periscope to lose control. You've got two possible solutions: either use the **/T** command to force tracing of the offending interrupt or use the **GA** command to force tracing of all interrupts.

Borland's **SIDEKICK** can interfere with the tracing of Interrupt 21H when **GT** is used. Force tracing of INT 21 using the **/T** command; use the **GA** command instead of **GT**; or better yet, remove **SIDEKICK** while debugging.

### Examples:

**GT PRINTLINE NEWPAGE** sets temporary code breakpoints at the addresses equal to the symbols **PRINTLINE** and **NEWPAGE**.

**GT** begins execution of the program with no temporary code breakpoints, only sticky code, debug register, and monitor breakpoints that are enabled.

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

## Command: Hex arithmetic (H)

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

---

**Description:** This command is used to perform hexadecimal arithmetic.

Addition, subtraction, multiplication, and division are available. The standard <arithmetic operator>s are used for each function. Each <number> must be in hex and may be from one to four hex digits. 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.

See also the X (translate) command.

**Examples:**

H 1234/123 gives an answer of 0010 R 0004.

H 1234\*123 gives an answer of 0014 B11C.

## **Command (Model IV): Hardware breakpoints All (HA)**

**Syntax:** HA [?] [\*] [+] [-]

**Description:** This command is used to display, clear, enable, or disable the hardware breakpoints.

HA ? displays all hardware breakpoints, HA \* clears all hardware breakpoints and display options, HA + enables all hardware breakpoints, and HA - disables all hardware breakpoints. If the hardware controls, HC, are set to the default values, nothing is displayed for them when HA ? is used. See the BA command for software breakpoints.

---

**NOTE:** Since RUN disables the breakpoints, use this command to re-enable the previously-set breakpoints.

**Examples:**

HA ? or HA displays all hardware breakpoints.

HA + enables all currently-set hardware breakpoints.

## Command (Model IV): Hardware Bit breakpoint (HB)

**Syntax:** HB [?] [\*] [+] [-]; or  
HB ?B XXXX XXXX for bytes where ? is L, M, N, or H; or  
HB ?W XXXX XXXX XXXX XXXX for words where ? is L, M, H, or blank; or  
HB ?3 XXXX XXXX XXXX XXXX XXXX XXXX where ? is L or H if 80386; or  
HB DW XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX for doublewords if 80386

**Description:** This command is used to display, clear, enable, disable, or set the hardware data bit breakpoint.

The bit breakpoint is entered as a binary number of exactly eight, 16, 24, or 32 characters for byte, word, three-byte, and doubleword values respectively. The allowable bit values are 0, 1, and X which correspond to a zero, one, and 'don't care' respectively. Only one breakpoint may be specified. The breakpoint may be used with or without an HM or HP breakpoint. If it is used with an HM or HP breakpoint, the bit breakpoint is logically ANDed with the memory or port breakpoint.

See the table in Figure 15-3 for the possible type codes.

HB ? displays the bit breakpoint, HB \* clears the bit breakpoint, HB + enables the bit breakpoint, and HB - disables

---

the bit breakpoint.

### Examples:

HB LB 1XXX XXXX sets the bit breakpoint for low byte accesses of any value containing a binary one in bit 7 and any value in bits 6 through zero.

HB HW 1010 01X0 0000 0000 sets the bit breakpoint for the value A400H or A600H in the high word.

## Command (Model IV): Hardware Controls (HC)

**Syntax:** HC [?] [\*] [#<number>] [B-|B+|B!]  
[C-|C+|C!] [O-|O+|O!] [P-|P+|P!]  
[S-|S+|S!] [TB|TC|TT] [X<byte>]

**Description:** This command is used to display, clear, or set the hardware controls. The controls include the pass count (#), buffer capture (B), cycle count capture (C), trace overflow stop (O), probe triggering (P), selective capture (S), trigger location (Tx), and exclude state(s) (X).

The pass count is entered as # followed by a <number> from one to FFFFH. It is used as a real-time breakpoint counter. When the specified number of breakpoints have occurred, Periscope interrupts the executing program. If one is specified as the pass count, the first breakpoint will interrupt the program. If the pass count is set to four, the fourth breakpoint will interrupt the program, etc. Note that the pass count must be used in conjunction with an HB, HD, HM, or HP breakpoint.

The buffer capture is enabled using B+ (default), disabled using B-, or toggled using B!. This control should be viewed as a master switch for buffer capture. When it is disabled, no information is added to the trace buffer regardless of other settings. Assuming you've captured some information in the buffer, but cannot save it to a disk file since DOS is busy, you could enter HA -;HC B-;GH {0:28\*4 to disable all hardware breakpoints, turn buffer



---

capture off, and go with hardware enabled to the next invocation of INT 28H, where it would be possible to write the trace buffer to a file. Also, you can write the trace buffer directly to a floppy disk without using DOS by entering **HW A:** or **HW B:**.

The cycle count capture is enabled using **C+**, disabled using **C-** (default), or toggled using **C!**. This control is used to force the generation of CPU cycle count records in the trace buffer when the cycle count overflows. It should be used for counting CPU cycles when selective capture is enabled with **S+**. Otherwise, leave this control in its default setting.

The trace overflow stop is enabled using **O+**, disabled using **O-** (default), or toggled using **O!**. When enabled, the overflow stop generates a breakpoint each time the hardware trace buffer fills up. This breakpoint allows you to see the CPU events in 2K groups, so you can examine program flow starting at a particular point. This control does not require a breakpoint to be set.

**NOTE:** To force the entire trace buffer to be viewable with the **HR**, **HT**, or **HU** commands even when the buffer is 'empty', use **HC O+**.

The probe triggering is enabled using **P+** (default), disabled using **P-**, or toggled using **P!**. This control is used to disqualify probe cycles from causing hardware breakpoints and to suppress display of the probe cycles, except in raw (**HR**) mode. When **HC P-** is used, any trace buffer cycle that shows **Probe** will not cause a hardware breakpoint, even if the address and/or data values associated with the CPU cycle would have otherwise generated a breakpoint.

This is needed for systems such as the IBM PS/2 Model 50, where RAM refresh cycles are captured in the Model IV trace buffer. These 'garbage' cycles can be very confusing. To filter them out, set jumper J5 on the pod so that it is connecting pins 1 and 2. In this position, the CPU Hold Acknowledge signal will be shown as **Probe** in the trace buffer. Then use **HC P-** to suppress display of these **Probe** records when viewing the trace buffer, except when the raw (**HR**) mode is used.

---

**NOTE:** Debugging programs running on IBM PS/2s with the Micro Channel bus architecture requires Periscope's active remote mode debugging feature, available separately.

The selective capture is enabled using `S+`, disabled using `S-` (default), or toggled using `S!`. When enabled, only events that would cause a hardware breakpoint are saved in the hardware trace buffer. This mode should be used with a pass count that indicates the number of events desired before the Periscope screen is displayed. For example, if you want to capture the next 16 Outs to port 3B4H, set the port breakpoint and enter `HC* #10 S+` to clear the controls and display options and then set them to capture only the next 10H trigger events. Finally, enter `GH` to arm the board and begin execution.

The selective capture of an event normally stops after `n` events have been collected, where `n` is the pass count setting. A special case of selective capture that allows continuous tracing is also supported. Simply set the pass count to `FFFH` and turn selective trace on. You'll have to stop the system using the break-out switch or some other method, but only the selected events will be in the trace buffer.

**NOTE:** When selective capture is used, not all CPU events are saved in the trace buffer, so it is not usually meaningful to disassemble the trace buffer.

Also, when `S+` is used, the trigger location is set to the bottom of the buffer. The cycle count field is usually valid only if the cycle count capture is on (using `C+`).

The trigger location may be set to the Top, Center, or Bottom of the trace buffer using `TT`, `TC`, or `TB` (default) respectively. When set to the top, the trace buffer shows 2K events after the trigger event, numbered from 0 to 7FE. When set to the center, the trace buffer shows up to 1K events before the trigger event and 1K events after the trigger event, numbered from -3FF to +3FF. When set to the bottom, the trace buffer shows up to 2K events before the trigger event, numbered from -7FE to 0. `TC` and `TT` are meaningful only

---

for a hardware breakpoint, not when the break-out switch is pressed.

**NOTE:** The trigger location that is in effect when a hardware breakpoint occurs controls the contents of the trace buffer. Although the trigger location may be changed after a breakpoint, the contents of the trace buffer remain the same.

The exclude state is used to exclude data from any or all sequential trigger states from being captured in the trace buffer. It is entered as HC X<byte>, where the <byte> may be any state from 0 to 7. For example, to exclude state 0 from the trace buffer, enter HC X0.

In this situation, only system activity while the sequential trigger is in states other than 0 is saved in the trace buffer. The excluded states may be cleared by entering HC X \*.

#### **Examples:**

HC \* #5 TT clears the hardware controls, sets the pass count to 5, and sets the trigger location to the top of the buffer.

HC O+;GH sets the trace overflow stop on and begins execution with hardware breakpoints enabled. After 2,048 (2K) CPU events have been added to the hardware trace buffer, control is returned to Periscope.

HC S+ #20 TC sets selective capture on with a pass count of 20H. Since selective trace is on, the TC command is ignored and TB is assumed.

HC #10 C+ S+ sets the pass count to 10H and turns cycle counting and selective capture on.

HC B- turns the buffer capture off.

---

## Command (Model IV): Hardware Data breakpoints (HD)

**Syntax:** HD [?] [\*] [+] [-]; or  
HD ?B <byte> <byte> for a byte range where ? is L,  
M, N, or H; or  
HD ?W <number> for a word breakpoint where ? is L,  
M, H, or blank; or  
HD ?3 <byte> <byte> <byte> for a 3-byte break-  
point, where ? is L or H if 80386; or  
HD DW <address> for a doubleword breakpoint if 80386

**Description:** This command is used to display, clear, enable, disable, or set hardware data breakpoints.

The data breakpoints are entered as a range of byte values for the byte breakpoints; single specific values for the word breakpoints; three bytes for the three-byte breakpoints; and an address for the doubleword breakpoints. Up to eight data breakpoints may be specified. This breakpoint may be used with or without an HM or HP breakpoint. If data breakpoints are used with an HM or HP breakpoint, the data breakpoints are ORed together as a group and then ANDed with the memory or port breakpoint.

See the table in Figure 15-3 for the possible type codes.

**NOTE:** The address entered for a doubleword breakpoint is handled in a special fashion. The segment portion of the address is used as the high part of the doubleword and the offset portion of the address is used as the low part of the doubleword. For example, entering  
HD DW 1234:5678 sets a breakpoint on the value 12345678H as a doubleword value.

After being set, these breakpoints are remembered until they are cleared.

Re-entering a previously set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the GH or GM command to make sure the breakpoints you've got are

---

the ones you want.

HD ? displays the data breakpoints, HD \* clears the data breakpoints, HD + enables the data breakpoints, and HD - disables the data breakpoints.

Setting data breakpoints can be tricky. Since Model IV watches the system from the vantage point of the CPU, it sees memory accesses as seen by the CPU, which is not necessarily the same as seen by the system bus. For example, if a word of 8-bit memory at B000:0 is read, the CPU sees it as one word read, but the bus sees it as two byte reads. If the instruction is a byte read rather than a word read, the CPU and the bus both see it as one byte read. For 16-bit memory, a word read at an even address is seen by the CPU and the bus as one word read.

---

DATA ACCESS TYPE	TYPE CODES	BYTES USED (HI TO LO)
80286:		
Word	W	XX
High byte	HB	X.
Low byte	LB	.X
80386:		
Doubleword	DW	XXXX
High 3 bytes	H3	XXX.
High word	HW	XX..
High byte	HB	X...
Low 3 bytes	L3	.XXX
Mid word	MW	.XX.
Next byte	NB	.X..
Low word	LW	..XX
Mid byte	MB	..X.
Low byte	LB	...X

---

*Figure 15-3. Data Access Types*

In an 80286 system, memory can be accessed as a byte or a word. How it is shown in the trace buffer depends on the access width and whether the starting address is even or odd (the address modulo 2). All memory accesses except word accesses starting at an odd address are completed in one CPU event. One CPU event may require multiple bus cycles.

In an 80386 system, memory can be accessed as a byte, word, three-byte, or doubleword. How the memory access is shown in the trace buffer depends on the access width and the starting address modulo 4. All memory accesses except the four cases shown in Figure 15-4 are completed in one CPU event. Again, one CPU event may require multiple bus cycles.

The tables in Figures 15-3 and 15-4 show the data access types and the CPU events required for various access widths and addresses on the 80286 and 80386 CPUs.

ACCESS WIDTH	ADDRESS (BINARY)	DATA ACCESS TYPE AND START ADDRESS
<b>80286:</b>		
Byte	xxxx0	Low byte (LB) at 0
Byte	xxxx1	High byte (HB) at 1
Word	xxxx0	Word (W) at 0
Word	xxxx1	High byte (HB) at 1; Low byte (LB) at 2
<b>80386:</b>		
Byte	xxx00	Low byte (LB) at 0
Byte	xxx01	Mid byte (MB) at 1
Byte	xxx10	Next byte (NB) at 2
Byte	xxx11	High byte (HB) at 3
Word	xxx00	Low word (LW) at 0
Word	xxx01	Mid word (MW) at 1
Word	xxx10	High word (HW) at 2
Word	xxx11	Low byte (LB) at 4; High byte (HB) at 3
Dword	xxx00	Doubleword (DW) at 0
Dword	xxx01	Low byte (LB) at 4; High 3 bytes (H3) at 1
Dword	xxx10	Low word (LW) at 4; High word (HW) at 2
Dword	xxx11	Low 3 bytes (L3) at 4; High byte (HB) at 3

*Figure 15-4. CPU Events for Various Access Widths*

Important points to note about the table in Figure 15-4 include:

- The physical width of the memory is invisible to the CPU. From the CPU, 8-bit, 16-bit, and 32-bit memory all respond in the same manner, if not at the same speed.
- Data breakpoints are complicated if BS-16 memory is

---

used in your 80386. See the file NOTES.TXT for more information.

- For even-word access to 8-bit memory, you won't see separate CPU events for the low and high bytes, just one word access.
- On an 80386 system, OUTs to ports 320H through 323H will show on the low, mid, next, and high bytes respectively. On an 80286 system, the above OUTs will show on the low, high, low, and high bytes respectively.
- Since everything is seen from the perspective of the CPU, memory moves are dictated by the code, not the physical width (8, 16, or 32 bits) of the memory device.
- Since some memory accesses may be split into multiple CPU events, it is important to take the address into account when setting data breakpoints.
- When using the H3 and L3 data breakpoints, specify three bytes of data from high to low addresses (to match the trace buffer display).
- When watching for writes of a character to the display, set the memory breakpoint as needed, plus the data breakpoints. Depending on the access method, you may need a byte (in any position), word (low or high), and/or doubleword breakpoint. The more you know about the conditions, the fewer data breakpoints you'll need.

**NOTE:** If you're not sure where to set a data breakpoint, try setting just the memory or port breakpoint and then run your program. After the breakpoint happens, look in the trace buffer to see how the memory or port was accessed. Then add the data breakpoint using the appropriate access type.

### Examples:

HD LB 10 1F allows breakpoints when the low byte is accessed and the value is from 10H to 1FH. A low word access of xx1F will not cause a breakpoint.

HD \* MW 1234 clears the data breakpoints and sets a breakpoint on the mid word with a value of 1234H.

---

HD DW { 0 : 10 \* 4 sets a doubleword breakpoint equal to the doubleword at INT 10H. This is a useful technique for setting a breakpoint on the next invocation of an interrupt in an 80386 system. This technique does not work on 80286 systems.

## Command (Model IV): Hardware Memory breakpoint (HM)

**Syntax:** HM [<address> <address> A|H|R|W|X  
[B] [(s,t)] [?] [\*] [+] [-] [...]

**Description:** This command is used to set, display, clear, enable, or disable memory breakpoints.

The breakpoints are entered as two addresses or a range. Up to eight breakpoints may be specified. The source of the breakpoint is indicated by an A, H, R, W or X, indicating interrupt Acknowledge, CPU Halt, memory Read, memory Write, and code prefetch (instruction eXecution) respectively.

The optional qualifier B is used to cause the breakpoint to capture information in the trace buffer and NOT to generate a breakpoint. When this qualifier is used, HC S+ is set automatically by Periscope. When using this qualifier, be sure to use another breakpoint to generate a trigger to stop execution of your program.

The optional argument (s,t) is used to set sequential triggers. S is the start state, from 0 to 6. T is the state that is moved to when the breakpoint condition is true, from 0 to 6 or ! to cause a trigger. See the section on sequential triggers below for more details.

Do not attempt to set memory breakpoints inside Periscope's code/data area. Also, watch out for breakpoint boundaries. A breakpoint set at B800:1 won't be triggered by a word or doubleword access at B800:0.



---

The code prefetch breakpoint is generated when an address is read into the prefetch queue, not when it is actually executed. Since the prefetch queue is six bytes long on the 80286 and 12 or 16 bytes long on the 80386, instructions are read before they're actually executed. Due to the 'breakpoint overrun' phenomenon, a breakpoint does not occur immediately. So the instruction in question may or may not have been executed by the time Periscope is activated.

If the desired instruction has not been executed, try setting the execution breakpoint after an instruction that flushes the prefetch queue (JMP, CALL, RET, IRET, INT, etc.).

After being set, these breakpoints are remembered until they are cleared. Re-entering a previously set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the GH or GM command to make sure the breakpoints you've got are the ones you want.

In an 80386 system, a breakpoint on code prefetch that is not on a doubleword boundary is rounded down to the nearest doubleword boundary, since all prefetch occurs on a doubleword boundary.

The standard segmented address format is used to set breakpoints in the first megabyte. To set breakpoints beyond 1MB, up to 16MB, use absolute addresses of four digits, an ampersand (&), and the remaining four digits.

HM ? displays the memory breakpoints, HM \* clears the memory breakpoints, HM + enables the memory breakpoints, and HM - disables the memory breakpoints.

**Sequential Triggers.** The sequential trigger capability is quite powerful. For example, to watch for the writing of the variable BAR while the routine FOO is executing, use the following sequence of commands:

```
HM FOO_START L1 X (0,1)
HM FOO_END L1 X (1,0)
```

---

HM BAR L2 W (1,1)

When the board is armed, it always starts in state 0. The first breakpoint advances from state 0 to state 1 on entry to FOO at FOO\_START. On exit from FOO at FOO\_END, the state is reverted from state 1 to state 0 by the second breakpoint. The third breakpoint generates a trigger if BAR is written while state 1 is active. If BAR is written while any other state is active, no trigger occurs.

Some restrictions apply:

- If the pass count or data breakpoints are used, they always apply to the last (highest numbered) state.
- State 7 is reserved for use by Periscope.
- If a true state is specified but not used somewhere as a current state, an error is displayed. For example, if (1,5) is used as a state setting, the true state 5 must be used as the current state somewhere, i.e., (5,x).
- Be careful when using fetches to switch states, since the CPU pipelining can cause fetches to occur much earlier than the memory activity performed by the instructions.
- The same trigger cannot be used in two different states.
- If HC TT is set, states 6 and 7 are reserved for use by Periscope.

### Examples:

HM B000:0 L1 W sets a breakpoint on writes to memory at B000:0 (the upper left-hand corner of the monochrome screen) for a length of one byte.

HM B000:1 L1 W sets a breakpoint on writes to memory at B000:1. This breakpoint is likely to never get a hit, since most access to display memory is done on word boundaries! If in doubt, extend the breakpoint to the next lower word or doubleword boundary. You may get some false hits, but at least you won't miss any!

HM 0:0 0:3FF R sets a breakpoint on reads of the interrupt vector table.

HM \* B800:0 L2 W (0,1);HM 0:46C L2 W

---

( 1 , 1 ) clears breakpoints and then sets a breakpoint on writes to memory at B800:0 for a length of 2. When this write occurs, the state is advanced to 1, which then enables the memory breakpoint on writes to 0:46C for a length of 2. When this breakpoint occurs, Periscope's screen is displayed.

HM 0:0 CS:0 W sets a breakpoint on writes to memory from the beginning of memory to the current code segment.

HM 10&0000 FF&FFFF X sets a breakpoint on execution in extended memory from the one megabyte point up to the 16 megabyte point.

## Command (Model IV): Hardware Port breakpoint (HP)

**Syntax:** HP [<port> [<port>] I|O [B]  
[(s,t)] [?] [\*] [+] [-] [...]]

**Description:** This command is used to set, display, clear, enable, or disable port breakpoints.

The breakpoints are entered as a single port or as a range of two ports, where each value must be from zero to FFFFH. Up to eight breakpoints may be specified. The source of the breakpoint is indicated by an I or O indicating an In (port read) or Out (port write) respectively.

The optional qualifier B is used to cause the breakpoint to capture information in the trace buffer and NOT to generate a breakpoint. When this qualifier is used, HC S+ is set automatically by Periscope. When using this qualifier, be sure to use another breakpoint to generate a trigger to stop execution of your program.

The optional argument (s,t) is used to set sequential triggers. S is the start state, from 0 to 6. T is the state that is moved to when the breakpoint condition is true, from 0 to 6 or ! to cause a trigger. See the section on sequential triggers

---

under the **HM** command for more details.

After being set, these breakpoints are remembered until they are cleared. Re-entering a previously-set breakpoint clears the breakpoint and displays the message **Breakpoint cleared**. Be careful to display all breakpoints before using the **GH** or **GM** command to make sure the breakpoints you've got are the ones you want. Do not attempt to set port breakpoints inside Periscope's port range.

**HP ?** displays the port breakpoints, **HP \*** clears the port breakpoints, **HP +** enables the port breakpoints, and **HP -** disables the port breakpoints.

#### **Examples:**

**HP 308 I** sets a breakpoint on reads of port 308H.

**HP 310 31F O** sets a breakpoint on writes of ports from 310H to 31FH.

## **Command (Model IV): display Hardware buffer in Raw mode (HR)**

**Syntax:** **HR [\*]** or **HR [\$] [! [#<number>]]**  
**[<segment>] [<file>]**

**Description:** This command is used to display the real-time trace buffer in a 'raw dump' format. Each line of the display corresponds to one CPU event. Each line contains an address, data, an operation type, a symbol corresponding to the address if available, the CPU cycle count from the previous trace buffer record, a sequence number, and possible other information.

The trace buffer may be cleared by entering **HR \***. The entire buffer may be displayed using **HR !**. Use the optional **#<number>** syntax to start the full buffer display at a specific sequence number. If an asterisk or exclamation point is not used, a full-screen display mode is entered un-

---

less the buffer is empty. The only way to exit this mode is to press the Esc key. The optional `<segment>` is used to decode addresses. The optional `<file>` indicates a trace buffer file previously created with the HW command that is used instead of the current contents of the trace buffer. To view the trace buffer starting where you were last time, use HR \$.

**NOTE:** When reading trace buffer information from a file, a decode segment must be specified.

**Layout of the Trace Buffer Display (see Figure 15-6).** The first field in the trace buffer is the address. It may appear in one of four formats: a segmented address (the segment, a colon, and one offset); a four-digit port number; a five-digit non-segmented address; or an eight-digit absolute address (four digits, an ampersand, and four digits).

The segmented address is displayed for data references when the address can be decoded into one of the following segments:

- User-specified data segment (set with D `<segment>`)
- User-specified code segment (set with S `<segment>`)
- A000, B000, C000, D000, E000, or F000 (BIOS segments)
- Periscope's segment
- DOS's segment

For code references, the decode order is:

- User-specified code segment (set with S `<segment>`)
- A000, B000, C000, D000, E000, or F000 (BIOS segments)
- PSP segment when RUN was last used
- Current derived segment (calculated from RETF and IRET instructions)
- Periscope's segment
- DOS's segment

The optional segment that can be entered on the command line overrides the user-specified code segment. To change the decode segment for data references while viewing the trace buffer, enter D `<segment>`. To change the decode segment for code references, enter S `<segment>`.

---

The four-digit port number is used for I/O port access. The five-digit non-segmented address is used when Periscope is unable to decode the segment into one of the above values. If addresses above one megabyte occur, the eight-digit absolute address is displayed.

The **second field** is the data for the CPU event. Depending on the CPU used and the method of memory access, the data field may be a byte, word, three-byte, or doubleword. See the tables in Figures 15-3 and 15-4 for more information on the possible data types.

Up to four bytes are shown, from high to low, in fixed columns. The address shown is for the lowest (rightmost) byte.

The **third field** is the operation. The possible values are:

- Fetch (code prefetch)
- Halt (CPU halt)
- In (I/O port read)
- Int Ack (interrupt acknowledge)
- Out (I/O port write)
- Probe (the probe line is high)
- Read (memory read, not including code prefetch)
- Cycle (cycle count overflow record generated when HC C+ is used)
- Write (memory write)
- Void (invalid trace buffer record)

All but the Probe operations are mutually exclusive. A symbol name may follow the operation if the address indicates a symbol.

**NOTE:** When displaying the hardware trace buffer, absolute addresses are used for symbol lookup so that as many symbols as possible are shown. If a symbol is found, the decode segments are changed as needed to match the segment of the symbol.

The **fourth field** is the CPU cycle count since the previous

---

trace buffer record. The cycle count may be shown as two, four, or five hex digits in brackets. If an illegal value is found, ? is displayed after the value. If you see an illegal value, please call Tech Support. Overflow records may display a plus sign following the cycle count. This can legitimately happen when too many CPU cycles occur between trace buffer records and HC C- is set.

To count the cycles from one point in the trace buffer to another, position the first item at the top of the display and then use # plus a sequence number or a search command to move to the second item. The accumulated cycle count is displayed in the cycle count field. Note that the number is in hex.

---

CPU SPEED (MHz)	TIME PER CYCLE (ns)
8	125
10	100
12.5	80
16	62.5
20	50
25	40

---

*Figure 15-5. Cycle Time by CPU Speed*

The 8-bit cycle count field can count up to 127 cycles per trace buffer record, which is enough for most continuously captured instructions. If you're using selective capture and want to count the CPU cycles between events, you'll need to turn the cycle capture on to force the capturing of cycle records. To do this, enter HC C+. The resulting records have an operation type of cycle and are displayed only when the HR mode is used.

When the HT mode is used, cycle records are not displayed and their cycle counts are accumulated and displayed on the next non-cycle record. With cycle capture on, the maximum elapsed time in the trace buffer for a 25 MHz system is 2K events times an average of 123 CPU cycles per trace buffer record times 40 nanoseconds per CPU cycle, or ap-

---

proximately .01 seconds.

To convert CPU cycle counts to time, use the table in Figure 15-5.

The **fifth field** is a sequence number. The hex number may be from -FFE to +FFE, depending on the trigger location. The first entry in the circular buffer is indicated by **Top** after the sequence number. Use the Home key to get to the top of the buffer. The last entry in the buffer is indicated by **Bottom** after the sequence number. Use the End key to get to the bottom of the buffer. Use the Left Arrow key to get to the center of the buffer (actually, 1K from the end of the buffer).

The **sixth field** follows the sequence number. If the address is within Periscope's code/data area, **PS** is displayed to identify the code/data as being Periscope. You may see these **PS** entries at the beginning and end of the trace buffer. They show where Periscope was turning control over to the application and regaining control from the application respectively.

A sample hardware trace buffer display in raw format is shown in Figure 15-6.

---

15C4:0138	A100	17EB	Fetch	START	[08]	-07B
15C4:013C	10BF	0134	Fetch		[02]	-07A
15C4:0140	0024	E001	Fetch		[02]	-079
15C4:FFFF		013B	Write		[04]	-078
15C4:0150	06B1	20CD	Fetch	DOSRET	[02]	-077
15C4:0154	8B00	02BE	Fetch	SAMPLE#45	[02]	-076
15C4:0158	A3E8	D304	Fetch		[02]	-075
15C4:015C	C88C	0134	Fetch		[02]	-074
15C4:0160	C32B	E8D3	Fetch	SAMPLE#52	[03]	-073
15C4:0002	A000		Read		[03]	-072
15C4:0164	C301	36A3	Fetch	SAMPLE#54	[04]	-071
15C4:0134		0280	Write	TOTMEM	[03]	-070
15C4:0168	B920	B050	Fetch	CONVERT	[02]	-06F
15C4:016C	AAF3	0003	Fetch		[02]	-06E
15C4:0136	0229		Write	FREMEM	[05]	-06D
15C4:FFFF		013B	Read		[02]	-06C

---

*Figure 15-6. Hardware Trace Buffer in Raw Format*

**Controlling the trace buffer display.** While in full-screen



---

mode, the positioning keys available are: Home, End, Up, Dn, PgUp, PgDn, and Left Arrow. While viewing the buffer, enter ? to display help on the functions available or press Alt-M to activate the menu system. There are a large number of commands used to control the display and to search through the trace buffer. They include:

- You can switch between buffer display modes by entering R (Raw), T (Trace), or U (Unassembly). To switch to Assembly-only mode (like the UA command), enter A. To switch to Both mode (like the UB command), enter B.
- You can move to any location in the buffer by entering # followed by the sequence number. Note that the sequence number must be from -7FE to +7FE. Any value outside this range is ANDed with 7FFH.

There are several toggles that control how the trace buffer is displayed (all of these are cleared when HA \* or HC \* is used). They are:

- The double quote (") is used to toggle between source-only and mixed (both source and assembly) modes. Mixed mode is the default.
- The letter C is used to turn the CPU cycle count display on and off (default) for disassembled lines. When using this mode, be aware that the cycle count allocation is not exact, since the cycle count is for the fetch of memory, which occurs before the instruction is executed. Also, since a source line may overlap the cycle count field, you may need to use the assembly-only mode to avoid conflicts.
- The letter E is used to toggle the display of extended memory on (default) and off. When the toggle is off, memory beyond 1 megabyte (an address of 0010&0000 or higher) is not displayed.
- The asterisk (\*) is used to toggle the display of flushed instructions. The default is to not display flushed instructions in the HT or HU mode. When the asterisk is pressed, flushed instructions are displayed, preceded by an asterisk. Flushed source lines show an asterisk instead of the usual colon after the line number.

When the prefetch queue is flushed on 80386 systems,

---

the next prefetch always starts at a doubleword boundary, regardless of where the instruction actually starts. This feature of the 80386 can confuse things, since Periscope tries to disassemble memory starting at the first byte and the instruction may actually start at the first, second, third, or fourth bytes. To manually control the start byte for the first instruction shown in the trace buffer display, enter a number from 0 to 3 to skip the same number of bytes for the first instruction only. After the first instruction, Periscope tracks instructions such as CALL, JMP, RET, RETF, INT, and IRET, and automatically skips the correct number of bytes each time the prefetch queue is flushed. It does not correctly track JMPs to registers (e.g. JMP AX), LOOPS or conditional JMPs.

- When extended memory is used, Periscope displays the address as an 8-digit absolute address. It is possible to fixup the address by entering `F <address> <address>`, where the first address is the address that is to be fixed up and the second address is the value that the first address is to be adjusted to. For example, `F D000:0 3000:0` fixes up the address D000:0 to be displayed as 3000:0 and also makes the same relative fixup for any address greater than or equal to D000:0. The second address must be less than the first address.

You can also use this command to fixup addresses beyond one megabyte. For example, `F 12&3456 FOO` fixes up 12&3456 to the address of the symbol FOO.

While the trace buffer display is active, you can also search for items in the trace buffer.

- To search for an address, enter `/A address`. To search for a range of addresses, enter `/A <range>`. If you're searching for references to B000:1, a word access to B000:0 would access the desired location, but show in the trace buffer as B000:0. If you get an unexpected hit when searching in HT or HU mode, switch to HR mode to see the record that the match occurred on. To search for addresses beyond one megabyte, use the absolute form of the address, `xxxx&xxxx`.
- To search for data values on an 80386 system, enter `/D`

<byte> <byte> <byte> <byte>. For wildcard bytes enter a question mark. On an 80286 system, use just the first two byte fields.

COMMAND	DESCRIPTION
# <number>	Move to record
/A <addr> <range>	Search for Address
/D <byte> <byte> <byte> <byte>	Search for Data
/F <type>	Filter type
/T <type>	Search for Type
0	Skip 0 bytes for first instruction
1	Skip 1 bytes for first instruction
2	Skip 2 bytes for first instruction
3	Skip 3 bytes for first instruction
"	Toggle unasm dupe display
*	Toggle flushed prefetch display
A	Use Asm mode
B	Use Both mode
C	Toggle cycle count display
D <segment>	Decode to Data segment
E	Toggle extended memory display
F <address> <address>	Fixup address
R	Use HR display
S <segment>	Decode to code Segment
T	Use HT display
U	Use HU display

Figure 15-7. Summary of Model IV Trace Buffer Commands

- To filter the display for records of a single type, enter /F x, where x is A (int Ack), H (CPU Halt), I (In), O (Out), P (Probe bit on), R (memory Read), W (memory Write), or X (code prefetch). This command forces HR mode. It is not possible to filter on cycle count or void records. Use of the type search cancels any filtering.
- To search for an operation type, enter /T x, where x is A (int Ack), H (CPU Halt), I (In), O (Out), P (Probe bit on), R (memory Read), W (memory Write), or X (code prefetch). The buffer search starts at the second line from the top of the screen. It is not possible to search for cycle count or void records.

### Examples:

HR displays the last page of the trace buffer.

HR 1234 decodes as many addresses as possible using 1234 as the segment value.

---

**HR \*** clears the hardware trace buffer.

**HR CS PSBUF.DAT** uses CS as the decode segment and reads the file PSBUF.DAT for the trace buffer information.

## **Command (Model IV): display Hardware buffer Single entry (HS)**

**Syntax:** HS [<segment>]

**Description:** This command displays the breakpoint event in the real-time hardware trace buffer in a 'raw dump' format.

HS displays the breakpoint event in the hardware trace buffer and then displays the Periscope prompt. It is used to display the CPU event that caused a hardware breakpoint. The display output is identical to that shown by the HR command. See the HR command for more information.

### **Example:**

GH; HS enables hardware breakpoints and displays the breakpoint event in the real-time hardware trace buffer after a hardware breakpoint occurs.

## **Command (Model IV): display Hardware buffer in Trace mode (HT)**

**Syntax:** HT [\*] or HT [\$] [! [#<number>]]  
[<segment>] [<file>]

**Description:** This command is used to display the real-time hardware trace buffer in a disassembly-and-data format. Code prefetches are disassembled in source or symbolic form. Other CPU events (read, write, in, out, etc.) are dis-

---

played in the 'raw' format described in the HR command.

The trace buffer may be cleared by entering HT \*. The entire buffer may be displayed using HT !. Use the optional # <number> syntax to start the full buffer display at a specific sequence number. If an asterisk or exclamation point is not used, a full-screen display mode is entered unless the buffer is empty. The only way to exit this mode is to press the Esc key. The optional <segment> is used to decode addresses. The optional <file> indicates a trace buffer file previously created with the HW command that is used instead of the current contents of the trace buffer. To view the trace buffer starting where you were last time, use HT \$.

**NOTE:** When reading trace buffer information from a file, a decode segment must be specified.

Depending on what is found in the trace buffer, HT may show nothing. If this happens, enter R to switch to the HR mode.

**NOTE:** See the HR command for more information on the full-screen trace buffer and the HU command for a discussion of the disassembly display.

Any memory or port accesses performed by an instruction (read, write, in, or out) are shown one or more lines after the instruction itself. These memory or port accesses may be the only way to determine that an instruction was actually executed. Register information is not available in real-time, but it is possible to deduce register values, especially by looking at the results of MOV instructions and implicit or explicit PUSHes and POPs, such as when an INT or IRET instruction is executed. You can often deduce the DS and ES registers from instructions that manipulate memory, etc.

A sample hardware trace buffer display in trace format is shown in Figure 15-8.

---

```

#29: start: call getmem           ; get memory size
15C4:FFFC 0138 Write                [13] -078
#44:      mov cl,6                ; shift count
#45:      mov si,2                ; point to top of memory in psp
#46:      mov ax,lsil             ; get top of memory
#48:      shr ax,cl               ; convert to KB
#49:      mov totmem,ax           ; and save total memory
#51:      mov bx,cs               ; get current segment
#52:      shr bx,cl               ; convert to KB
#53:      sub ax,bx               ; subtract from total memory to get
15C4:0002 A000 Read                [0E] -072
#54:      mov fremem,ax           ; free memory
#55:      ret
15C4:0134 0280 Write TOTMEM         [07] -070
15C4:0136 0229 Write FREMEM        [09] -06D
15C4:FFFC 0138 Read                [02] -06C

```

---

*Figure 15-8. Hardware Trace Buffer in Trace Format*

### Examples:

HT 9000 displays the last page of the hardware trace buffer, showing all addresses from 9000:0 to 9000:FFFF in segmented format.

HT \* clears the hardware trace buffer.

Ctrl-P then HT ! dumps the entire trace buffer to the printer.

HT \$ displays the trace buffer starting at the same point you were the last time HT was used.

## Command (Model IV): display Hardware buffer in Unasm mode (HU)

**Syntax:** HU [\*] or HU [\$] [! [#<number>]]  
 [<segment>] [<file>]

**Description:** This command is used to display the real-time hardware trace buffer in a disassembly-only format. Code

---

prefetches are disassembled in symbolic form. Other CPU events (read, write, in, out, etc.) are not shown.

The trace buffer may be cleared by entering `HU *`. The entire buffer may be displayed using `HU !`. Use the optional `#<number>` syntax to start the full buffer display at a specific sequence number. If an asterisk or exclamation point is not used, a full-screen display mode is entered unless the buffer is empty. The only way to exit this mode is to press the Esc key. The optional `<segment>` is used to decode addresses. The optional `<file>` indicates a trace buffer file previously created with the `HW` command that is used instead of the current contents of the trace buffer. To view the trace buffer starting where you were last time, use `HU $`. When reading trace buffer information from a file, a decode segment must be specified.

Depending on what is found in the trace buffer, `HU` may show a blank display. If this happens, enter `R` to switch to the `HR` mode.

**NOTE:** See the `HR` command for more information on the full-screen trace buffer.

The Up and Dn keys move up or down by one record. This will not necessarily result in a change of the screen. Similarly, the PgUp key moves backward by as many records as there are lines displayed. The best results are obtained by using the PgDn key, since this key best preserves the alignment of the display.

If the instruction matches to a source line and DOS is available, the source line is displayed. The disassembled lines are in the standard disassembly format, showing an address, the opcodes making up the instruction, and the instruction itself. The address is decoded per the rules described in the `HR` command.

When disassembling instructions, one or more trace buffer records are required per instruction. All code prefetch records are added to a buffer which is then disassembled. Even though code has been fetched by the processor, there is no absolute method of telling whether the code was actually executed, so Periscope attempts to infer the execution

---

of code in the buffer. If a discontinuity in the address occurs or after an unconditional JMP, CALL, RET, RETF, INT or IRET instruction is found, the buffer is flushed, since the CPU's prefetch buffer acts in a similar fashion. Most problems occur with conditional jumps since Periscope can't determine whether the jump was taken.

Flushed instructions are not normally displayed. To toggle the flush display code, enter \* while viewing the trace buffer. Instructions that Periscope thinks were flushed will be shown with a leading asterisk. Press \* again to turn the flush display mode off.

When using the Up or Dn keys, the display can vary drastically. This is due to the cumulative effect of the buffered instructions and Periscope's flush logic. Instructions can disappear and then reappear in odd positions. You might think of this erratic movement as being like house of mirrors, where a slight movement can cause a large distortion in what you see. For best results, move one or more pages above the event you want to see and then page down into the area of interest. Also, consider using Ctrl-P and HT I or HU I to output the trace to a printer.

A sample hardware trace buffer display in disassembly (unasm) format is shown in Figure 15-9.

---

#29:	start:	call getmem	; get memory size
#44:		mov cl,6	; shift count
#45:		mov si,2	; point to top of memory in psp
#46:		mov ax,[si]	; get top of memory
#48:		shr ax,cl	; convert to KB
#49:		mov totmem,ax	; and save total memory
#51:		mov bx,cs	; get current segment
#52:		shr bx,cl	; convert to KB
#53:		sub ax,bx	; subtract from total memory to get
#54:		mov fremem,ax	; free memory
#55:		ret	

---

*Figure 15-9. Hardware Trace Buffer in Unasm Format*

### Examples:

HU S.START displays the last page of the hardware trace



---

buffer, decoding all possible addresses to the segment of the symbol START.

HU \* clears the hardware trace buffer.

Ctrl-P and HU ! #-200 dumps the trace buffer starting at sequence number -200 to the printer.

## Command (Model IV): Hardware Write (HW)

**Syntax:** HW <file> or HW A: or HW B:

**Description:** This command is used to save the contents of the hardware trace buffer to a disk <file> or to a floppy disk. The saved trace buffer can be viewed with the HR, HT, and HU commands.

When the first form of the command is used, DOS is used to write the <file>. The file created is always 20,490 bytes in length, which is the 20,480-byte trace buffer plus a 10-byte header.

The last two forms of this command use BIOS calls only (INT 13H) and therefore do not require DOS to be available. If you want to write the buffer to a floppy disk, a previously-formatted floppy must be in the specified drive (A: or B:). Tracks 27 through 32 on head 0 are used. No DOS directory support is provided, so any existing files may be partially or completely overwritten! To create a DOS-readable file, you must then run PS4TEST with the /WA or /WB options to read the indicated floppy and write the buffer file PSBUF.DAT.

### Examples:

HW PSBUF.DAT saves the current contents of the hardware trace buffer to the file PSBUF.DAT. To view the saved trace buffer, enter HT CS PSBUF.DAT, where CS is the "decode" segment.

---

**HW A:** saves the contents of the trace buffer to the floppy disk in drive A. To create a DOS-readable file, enter **PS4TEST /WA** at the DOS prompt. Then, from Periscope, enter **HT CS PSBUF.DAT** to display the trace buffer.

## Command: Input (I)

**Syntax:** **I** <port> or **IW** <port>

**Description:** This command is used to read an I/O port. The first form reads a byte while the second form reads a word.

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 value retrieved by reading the port is displayed on the line following the command.

### Examples:

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

**IW DX** performs a word read of the port indicated by register DX and displays the word input.

## Command: Interrupt Compare (IC)

**Syntax:** **IC**

**Description:** This command is used to compare the current value of the interrupt vectors with their previously-saved values.

**IC** is available only after an **IS** command has been used to

---

save the interrupt vectors. After an **IR** command has been used, this command is disabled until another **IS** command has been used.

Assuming the vector for Interrupt 0 had been changed, the display of the **IC** command might be **00 0294:588F→2345:6789**, where the first field is the interrupt number, which is followed by the old interrupt vector and the new (current) interrupt vector.

**Example:**

Assuming the interrupt vectors were previously saved using the **IS** command, enter **IC** to compare all vectors with their saved values.

## **Command: Interrupt Restore (IR)**

**Syntax:** **IR**

**Description:** This command is used to restore the interrupt vectors to a previously-saved state.

This command is usable only after an **IS** command has been used to save the interrupt vectors. After an Interrupt Restore has been performed, the Interrupt Restore command is disabled until another Interrupt Save has been performed.

**Example:**

Assume the interrupt vectors were previously saved using the **IS** command. Enter **IR** to restore all vectors to their values saved by the **IS** command.

---

## Command: Interrupt Save (IS)

**Syntax:** IS

**Description:** This command is used to save the interrupts for later comparison or restoration.

The Interrupt Save command saves the current state of the machine's interrupt vectors in case you need to restore the vectors to that state at some later point. For example, assume you're debugging a program that modifies some of the interrupt vectors. If you need to terminate execution of the program, you can restore the interrupt vectors and then use the QR command to return to DOS.

To use the Interrupt Save command, enter IS when the Periscope prompt is displayed. Later, you can restore the vectors to their saved state by using the IR command.

To prevent accidental restoration of the vectors, the IS command sets a flag that is cleared by the IR command. When this flag is cleared, the IR command generates an error.

**Example:**

Enter IS to save the interrupt vectors. At any point later, the IR command may be used to restore the vectors to their saved state.

## Command: Jump (J)

**Syntax:** J

**Description:** This command is used as a shorthand form of Go, to execute (step) to the next instruction.

It executes the current instruction at full speed, avoiding tracing through the execution of CALL, INT, LOOP or

---

other repeated instructions. 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 CALLs or INTs that do not return control to the next instruction.

### **Examples:**

Assume that the current instruction is INT 21. Enter J to place a temporary code breakpoint at the instruction after the INT 21 and execute the INT at full speed.

Assume that the current instruction is RET. Enter J to trace to the next logical (not physical) instruction.

## **Command: Jump Line (JL)**

**Syntax:** JL

**Description:** This command is used to step to the next source-code line.

The JL command performs Jump (step) commands until the next source line is found. This is a quick method of moving through a high-level language program, keeping to the source-code lines. If no source line symbols are found,

---

this command acts like the J command. See also the TL command.

**Example:**

Assume the current instruction is line 10 of the first source-code module. Enter JL to step to the next logical source-code line.

**NOTE:** If your compiler does not generate line numbers for every line, some lines may be skipped.

## **Command: clear (K)**

**Syntax:** K

**Description:** This command clears the Periscope screen and regenerates the windows if used. It has no arguments.

**Example:**

K clears the screen.

## **Command: clear and Initialize (KI)**

**Syntax:** KI

**Description:** This command initializes the monitor, clears the Periscope screen, and regenerates the windows if used. It has no arguments.

This variant of the clear command should be used if Periscope's screen is not in text mode on entry to Periscope. Do not use this command if 43-line mode is set. It will revert the screen to 25-line mode.

---

### Example:

KI performs a mode set and clears the screen.

## Command: Load Absolute disk sectors (LA)

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

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

The segment defaults to CS if no segment is specified in the <address>. The <drive> is either a single-digit number indicating the disk drive (0 = A, 1 = B, etc.) or A:, 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 not be busy. This command uses DOS interrupt 25H. See the DOS manual for information on the numbering of the absolute disk sectors.

### Examples:

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

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

## Command: Load Batch file (LB)

**Syntax:** LB <file>

---

**Description:** This command is used to load a batch or script file that contains Periscope commands.

To use this command, DOS must not be busy. The input file defaults to an extension of .PSB. The file may be created by the WB command, the /K command, or a text editor. Any legal Periscope commands may be placed in the file.

**Example:**

After using WB SAVE to save the breakpoint and window settings to the file SAVE.PSB, use LB SAVE to later reload the breakpoint and window settings.

## **Command: Load alias and record Definitions (LD)**

**Syntax:** LD \* or LD <file>

**Description:** This command is used to clear and/or load alias and record definitions from a PSD <file> created by RS.

The asterisk clears the alias and record definitions. The LD command cannot be used to load a DEF file. Only PSD files are supported. See the description of RS in Chapter 10. To load a file, DOS must not be busy.

**Examples:**

LD \* clears all alias and record definitions.

LD FTOC loads the alias and record definitions from the file FTOC.PSD.



---

## Command: Load File from disk (LF)

**Syntax:** LF [<address>]

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

The optional <address> specifies where the file is to be loaded. If the address is not specified, CS:100 is used. To use this command, DOS must not be busy. 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 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.

**NOTE:** This command cannot load into memory beyond the end of DOS memory. For a 640K system, the maximum address is 9000:FFFF.

### 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: Load Symbols from disk (LS)

**Syntax:** LS \* or LS <segment><file>

---

**Description:** This command is used to load a Periscope symbol file into the symbol table.

The asterisk clears the symbol table. The <segment> contains the relocation factor that is added to the segment values found in the PSS file. For COM files, this is the value of the PSP segment or CS. For EXE files, this is the value of the PSP segment plus 10H. To have Periscope calculate the correct value for an EXE file that is the currently active program, use \$ as a segment value. The <file> is the path and file name of a PSS file.

This command cannot be used to load a MAP file. Only PSS files are supported. A version check is done to ensure that the symbol file used is compatible with the current version of Periscope. To use this command, DOS must not be busy.

**Examples:**

LS \* clears the symbol table.

LS CS SAMPLE loads the file SAMPLE.PSS into the symbol table, relocating the segments by the current value of CS.

LS \$ FTOC loads the file FTOC.PSS into the symbol table at PSP + 10H for the program FTOC.EXE.

## Command: Move (M)

**Syntax:** M <range> <address>

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

The segment and offset must be specified for both addresses. 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).

**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).

**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 (N)

**Syntax:** N <file>

**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 using the **LF** and **WF** commands.

The <file> parameter is 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 for 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 semi-colon or a carriage return is found. If the PSP cannot be found, the **LF** and **WF** commands can still be used, presuming that DOS is not busy.

The Name command also requires that the PSP's address has been set by **RUN** 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.

## **Command: Output (O)**

**Syntax:** O <port> <byte> or OW <port> <word>

**Description:** This command is used to write to an I/O port. The first form writes a byte and the second form writes a word.

The <port> 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. The <word> value output may be from 0 to FFFFH.

**Examples:**

O 100 FF outputs FFH to port 100H.

OW DX 1234 outputs 1234H to the port indicated by register DX.

O DX AX returns an error since register AX represents a word. Use OW DX AX to output a word.

## **Command: Quit (Q)**

**Syntax:** Q, QB, QC, QL, QR or QS

---

**Description:** This command is used to display the reason Periscope was activated and to exit Periscope.

By entering Q, the entry reason is displayed. The possible entry reasons are:

BR - A monitor breakpoint was taken.

DR - An 80386 debug register breakpoint occurred.

GO - A code breakpoint was taken.

HW - A hardware breakpoint occurred (Periscope III or IV only).

P1 - Parity error 1 (motherboard) occurred.

P2 - Parity error 2 (expansion memory) occurred. This is normal when Periscope II's break-out switch is used.

SW - The break-out switch was pressed. This also appears for entry via hot keys set with PSKEY.

TR - An instruction was traced.

WD - A watchdog interrupt occurred on a PS/2 machine.

X6 - Exception interrupt 6 (illegal opcode) occurred.

XD - Exception interrupt 0DH (segment wraparound) occurred.

The QB (boot) command boots the system. It is the same 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. If an exception interrupt occurs, you should boot the system as soon as possible.

The QC (continue) command 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. Use the Go command if you want to set any breakpoints.

The QL (long boot) command puts the system through full diagnostics, as when the system is first powered on. It should be used if you need to reinstall any BIOS drivers such as the EGA or VGA.

---

The **QR** (return to DOS) command is used to abandon execution of the current program and return to DOS. Any open files will not be closed, which can cause problems if the files have been updated. Any changes the program has made to interrupt vectors will not be backed out. Also, the keyboard buffer is cleared. When possible, use **QC** or **G** instead of this command.

**NOTE:** The **QR** command is allowed only if **RUN.COM** was used to set the PSP for the program being executed. It also requires that DOS is not busy.

The **QS** (short boot) command 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. Since all interrupts are not restored, this boot option is more fragile than the other two.

Due to the unchanged vectors, some resident programs and drivers may think they're still installed. If you have problems, either quit using this option or use the user exit **F1** to clean up the necessary vectors on the way out.

**NOTE:** The short boot is not compatible with all systems. If it doesn't work in your system, try removing all device drivers and memory-resident programs. If it still doesn't work, there's probably not much hope.

There are dedicated user exits for the **QB**, **QL**, and **QS** commands. They enable you to customize the cleanup process for your system before rebooting. See the sample program **USEREXIT.ASM** for more information.

### **Examples:**

**Q** displays the entry reason.

**QC** exits Periscope and continues execution of the inter-

---

rupted program without setting any breakpoints.

QB exits Periscope and performs a normal boot.

## Command: Register (R)

**Syntax:** R [<register>] or R+ or R=<address>  
or R<byte> [<number>] or R?

**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 information is shown in the appropriate windows if windows are being used.

To modify the registers or flags, enter R xx, where xx is the register name. Use FL for the flags. To easily move to the next instruction, enter R+. Periscope disassembles the current instruction and sets IP to the start of the next instruction.

To change CS:IP to a new address, enter R=<address>.

There are ten user registers that you may use to hold any word value you desire. In the syntax shown above, the <byte> is a number from 0 to 9. For example, to hold the current value of SP in user register number one, enter R1=SP. Then at a later time, you could enter DW SS:R1 to use the saved user register. To display the contents of a single user register, enter just the register name. To display the contents of all user registers, enter R?. These registers can be used wherever a 16-bit register is usable.

In all of the examples shown in Figure 15-10, the first two lines display the current values of the registers and the flags.

See the <flag> command parameter in Appendix C for an explanation of the flag mnemonics. The last line in each of the examples in Figure 15-10 shows the disassembled instruction. The address of the instruction (CS:IP) is shown at the left, followed by the bytes that make up the instruction, and the instruction itself.

In Example 2 shown in Figure 15-10, 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 accesses before the execution of the instruction. The effective address calculations and displays for real-mode instructions are supported (up to the 80286 and the 80287), with two exceptions. 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, and the FRSTOR and FSAVE instructions do not show the 94 bytes they read and write.

---

```

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 FL=0046 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 FL=0246 NV UP EI PL ZR NA PE NC
WR DS:131 = 0000
063A:010E 891E3101 MOV [FILEOFFSET1],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 FL=0246 NV UP EI PL ZR NA PE NC
P565:
063A:01AD BF2F01 MOV DI,012F ; FILESEGMENT

```

---

*Figure 15-10. Sample Displays of Registers and Flags*

In Example 3 shown in Figure 15-10, 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.



---

If the current instruction is a conditional jump (see **Example 1** in Figure 15-10), 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** in Figure 15-10). 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** in Figure 15-10). 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 FILESEGMENT`.

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, FS, GS, SS, CS, IP, and FL (flags). The valid 8-bit register names are AH, AL, BH, BL, CH, CL, DH, and DL.

To modify a flag, enter `R FL`. Periscope displays the current values of the flags (see the `<flag>` parameter in Appendix C) 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. The hex value of the flag register is displayed following the

---

**mnemonic FL.**

Changes to registers and flags are highlighted, even for the software traceback display. This makes it easy to see what changes were made since the last time Periscope's screen was displayed.

When windows are used, the register display can be put into a vertical window. This format requires 18 lines to display all the registers and flags. If fewer lines are available, the register display is truncated. When the vertical window is used, the effective address is displayed in the separator line following the watch window. This mode may be toggled on and off by pressing the Alt-R key combination. You can also set vertical registers by using R: 1 in the windows specification. See the /W command for more information.

If you're using an 80386 or later system, a vertical display of the 80386 registers is also available. Press the Alt-3 key combination to toggle 386 registers on and off. You can also use R: 3 in the windows specification. Note that the high part of the 32-bit registers and the FS and GS registers are never highlighted when changed.

### **Examples:**

R displays all registers and flags, the effective address for reads and/or writes, and disassembles the current instruction. It also forces redisplay of the disassembly window if one is used.

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+ moves the instruction pointer to the beginning of the next instruction.

R= \_MAIN sets CS:IP to the symbol \_MAIN.

---

**R FL** 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 FL ZR**.

**R0=AX** saves the current value of register AX to user register R0.

**R0** displays the current value of user register R0.

**R?** displays the current value of all non-zero user registers.

## Command: Register Compare (RC)

**Syntax:** RC

**Description:** This command is used to compare the current value of the registers with their previously-saved values.

RC is available only after an **RS** command has been used to save the registers. After an **RR** command has been used, this command is disabled until another **RS** command has been used.

Assuming that register CX has been changed, the display of the RC command might look like that shown in Figure 15-11, where the first line shows the register names, the second line shows the values of the registers at RS time, and the third line shows the changed registers at RC time. Unchanged values are indicated by the double quote marks.

### Example:

Assuming the registers were previously saved using the **RS** command, enter **RC** to compare all registers with their saved values.

---

	AX	BX	CX	DX	SP	BP	SI	DI	DS	ES	SS	CS	IP	FL
ERS:	0000	0000	0000	0000	FFFE	0000	0000	0000	1382	1382	1382	1382	0100	0246
ERC:	''	''	''	''	FFFB	''	''	''	''	''	''	026A	143F	''

---

*Figure 15-11. Display of RC Command*

## Command: Register Restore (RR)

**Syntax:** RR

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

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

**Example:**

Assume the registers were previously saved using the RS command. Enter RR to restore all registers to their values saved by the RS command.

## Command: Register Save (RS)

**Syntax:** RS

**Description:** This command is used to save the registers for later comparison or 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. To prevent accidental restoration of the registers, the **RS** command sets a flag that is cleared by the **RR** command. If this flag is not set, **RR** generates an error.

### **Example:**

Enter **RS** to save the machine registers. The **RR** command may then be used to restore the saved register values, and the **RC** command may be used to compare register values with the saved values.

## **Command: Search (S)**

**Syntax:** **S** [**!x**] <range> <list>

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

The block of memory specified by the <range> is searched for the pattern specified by the <list>. If a match is found, the starting address and symbol name, if any, 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.

A wildcard search capability is available using the optional **!x** parameter shown above. The wildcard character **x** can be used in the list parameter to indicate a wildcard field. For example, to search for all occurrences of the byte string **EBH**, a wildcard, and **90H** (short jumps followed by a **NOP**), use **S !? CS:100 LCX EB ? 90**. Avoid using hex characters for the wildcard. Use **?** or a period when possible to minimize confusion. Note that the wildcard character need not be in quotes, but any other non-hex field

---

does.

This wildcard capability was originally inspired by a need to work around a weakness in MASM, the generation of unneeded NOPs. A better solution is now available. OPTASM from SLR Systems is a plug-compatible replacement for MASM that doesn't generate spurious NOPs and is four times faster, too!

### 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 PRINTLINE L 50 C "Page"** searches 50H bytes starting at the address of the symbol PRINTLINE for the byte 0CH followed by the string Page.

## Command: Search for Address reference (SA)

**Syntax:** SA <range> <address>

**Description:** This command is used to search memory for references to a specified address. It does not show any source code, just disassembled instructions.

This command can be thought of as a disassembly that only shows instructions that reference an address of interest. To use it, specify a <range> that is to be searched and the <address> that is to be searched for. If you're not using a symbol name for the address, 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 Jumps 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.

---

**NOTE:** References to stack data variables are not shown.

**Examples:**

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

**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 Calls (SC)**

**Syntax:** SC [<byte>]

**Description:** This command searches the stack for references to CALLED subroutines and software INTerrupts. If a match is found, the disassembly of the CALL or INT is displayed. This technique can help you determine the calling sequence used by a program and to unravel nested code. SC is similar to SR. SR analyzes the stack looking outward, while SC analyzes the stack looking inward. The optional <byte> field is used to override the default length of 10H stack entries.

The results are usually accurate, but cannot be guaranteed. For example, if a PUSH instruction saves a value on the stack that is the same as the address of the instruction after a CALL instruction, a false hit will occur.

This command does not interpret hardware interrupts since there is no interrupt in the instruction stream to indicate what happened.

**Example:**

SC searches the stack for CALLS and INTS. If any are found, the disassembled instruction is displayed. Note that

---

the most recent item is displayed first.

## Command: Search then Display (SD)

**Syntax:** SD <range> <list>

**Description:** This command is used to search memory for a byte/string pattern and then display the matches. This command is the same as the Search command, except that any matches are displayed in byte format. If a data window is used, the matching address is displayed in the active window. After a key press, the search continues. See the Search command for more information.

### Example:

SD CS:0 FFFF "Hello" searches memory from CS:0 to CS:FFFF for the string Hello. If any matches are found, they are displayed in byte format.

## Command: Search for Return address (SR)

**Syntax:** SR [<byte>]

**Description:** This command searches the stack for return addresses. Each stack item is examined to see if it contains an address after a CALL or INT instruction.

This command is similar to the SC command. SR analyzes the stack looking outward, while SC analyzes the stack looking inward. The optional <byte> field is used to override the default length of 10H stack entries.

If a match is found, the address of the instruction and the offset to the nearest symbol after the CALL or INT is displayed. This technique can help you determine the calling sequence used by a program and to unravel nested code. The results are usually accurate, but cannot be guaranteed.



---

For example, if a PUSH instruction saves a value on the stack that is the same as the address of the instruction after a CALL instruction, a false hit will occur.

Some programs may manipulate the stack in ways that cause this command to fail. For example, Lattice C can add an odd value to the SP register, which causes the stack to be viewed incorrectly.

This command does not interpret hardware interrupts since there is no interrupt in the instruction stream to indicate what happened.

**Example:**

SR searches the stack for return addresses. If any are found, the return address and the offset from the next lower symbol are displayed. The address shown is the instruction following a CALL (near or far) or an INT. Note that the most recent item is displayed first.

15

## **Command: Search for Unassembly match (SU)**

**Syntax:** SU <range> <list>

**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 a <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.

To find all occurrences of MOV SP, enter "MOV SP". Any lower case input is converted to upper case before the search is performed. The search starts in the mnemonic field and goes through the end of the argument field. Any blanks in the list field are ignored, so "MOVSP" is the same

---

as "MOV SP".

**NOTE:** This command will not find source-code lines or procedure labels, just disassembled instructions.

**Examples:**

SU CS:100 L 200 "MOV SS" searches from CS:100 for 200H bytes for any instructions that contain MOV SS.

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

## Command: Trace (T)

**Syntax:** T [<number>]

**Description:** This command is used to trace through the current program one instruction at a time (i.e., single step).

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 is traced. After each trace, the registers, effective address, and current instruction are displayed.

If you're using a single-monitor system to trace code that does not do any screen writes, you may want to use the Alt-N keys to turn the screen swap off. This eliminates the annoying flash caused by the program's screen being restored in case the instruction updates the screen.

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 / Trace Registers / Trace Unasm (TB/TR/TU)

**Syntax:** TB | TR | TU [\*] [! [#<number>]]

**Description:** These commands are used to view the software trace buffer. Do not confuse this buffer with the real-time hardware trace buffer available with Periscope III and IV.

The software trace buffer is used to save the machine registers each time Periscope is exited. This circular buffer can contain zero to 2016 entries, depending on the 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.

To display the entire trace buffer (to dump it to a file or the printer), use an exclamation point when starting the command. If you also enter the optional #<number>, the dump begins at the specified sequence number.

Once in the full-screen buffer, you can position the buffer with a #<number> entry, where <number> is the sequence number. If the number entered is too low, the first entry in the buffer is shown. Similarly, if the number

---

entered is too high, the last entry in the buffer is shown.

---

```
AX=0000 BX=0000 CX=0008 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000 #0001
DS=15E6 ES=15E6 SS=15E6 CS=15E6 IP=0137 FL=0346 NV UP EI PL ZR NA PE NC
START:
15E6:0137 E81700 CALL GETMEM

AX=0000 BX=0000 CX=0008 DX=0000 SP=FFFC BP=0000 SI=0000 DI=0000 #0002
DS=15E6 ES=15E6 SS=15E6 CS=15E6 IP=0151 FL=0346 NV UP EI PL ZR NA PE NC
GETMEM:
15E6:0151 B106 MOV CL,06

AX=0000 BX=0000 CX=0006 DX=0000 SP=FFFC BP=0000 SI=0000 DI=0000 #0003
DS=15E6 ES=15E6 SS=15E6 CS=15E6 IP=0153 FL=0346 NV UP EI PL ZR NA PE NC
15E6:0153 BE0200 MOV SI,0002
```

---

*Figure 15-12. Software Trace Buffer Using the TB Command*

Figure 15-12 shows three consecutive entries in the format used by the TB command.

The trace buffer may be cleared by entering **TB \***. If an asterisk is not used, a full-screen display mode is entered. The only way to exit this mode is to press the Esc key. The keys available are: Home, End, Up, Dn, PgUp, PgDn, and Esc. If you press any other key, the message **Press Esc to end full-screen mode** is displayed.

The TR and TU commands are subsets of the TB command. TR displays just the registers and sequence number, and TU displays just the disassembled instruction. The same records shown in Figure 15-12 are shown in Figure 15-13 in the format displayed by the TR command.

---

```
AX=0000 BX=0000 CX=0008 DX=0000 SP=FFFE BP=0000 SI=0000 DI=0000 #0001
DS=15E6 ES=15E6 SS=15E6 CS=15E6 IP=0137 FL=0346 NV UP EI PL ZR NA PE NC

AX=0000 BX=0000 CX=0008 DX=0000 SP=FFFC BP=0000 SI=0000 DI=0000 #0002
DS=15E6 ES=15E6 SS=15E6 CS=15E6 IP=0151 FL=0346 NV UP EI PL ZR NA PE NC

AX=0000 BX=0000 CX=0006 DX=0000 SP=FFFC BP=0000 SI=0000 DI=0000 #0003
DS=15E6 ES=15E6 SS=15E6 CS=15E6 IP=0153 FL=0346 NV UP EI PL ZR NA PE NC
```

---

*Figure 15-13. Software Trace Buffer Using the TR Command*

**NOTE:** Since Periscope adds to the buffer each time it

---

is exited, watch out for possible discontinuities in the software trace buffer. If you're using the T, GA, or GT commands, there's no problem. If you're using the G or J commands, not all instructions will be 'seen' by Periscope. The unseen instructions leave discontinuities in the trace 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. If the trace buffer is empty, a TB, TR or TU command has no effect.

When using the TB, TR, or TU commands to display the software trace buffer, you can switch among any of the three formats by keying B, R, or U, respectively.

#### **Examples:**

TB shows both the register and disassembly display of the software trace buffer.

TU shows just the disassembly display.

TB \* clears the software trace buffer and resets the sequence number to zero.

## **Command: Trace all but Interrupts (TI)**

### **Syntax: TI**

**Description:** This command is used to trace (single step) all instructions except interrupts.

If a software interrupt is the current instruction, this command sets a temporary code breakpoint on the next instruction and executes the interrupt at full speed.

---

**Example:**

If the current instruction at offset 120H is INT 21H, the **TI** command sets a temporary code breakpoint at offset 122H and executes to that point at full speed.

## **Command: Trace Line (TL)**

**Syntax:** TL

**Description:** This command is used to single step until the next source-code line is executed.

The **TL** command performs Trace commands until the next source line is found. This command can be used to get back to your source code if you come into Periscope and no source code is displayed. Since this command single steps the code while checking for a source line, it can be slow. In many situations, the **JL** command will be faster than this command. If no source line symbols are found, this command acts like the **T** command.

**Example:**

Assume the current instruction is in a library routine. Enter **TL** to single step your code until an instruction corresponding to a source line is executed.

## **Command: Unassemble memory (UA/UB/US)**

**Syntax:** U[A|B|S] [<range>]

**Description:** This command is used to disassemble memory. Memory is disassembled in the Assembly mode, source-and-assembly (Both) mode, or Source-only mode as set by the **UA**, **UB**, and **US** commands respectively. Source is the

---

default mode, unless the /E:0 installation option was used or no line numbers are found, in which case Source-and-assembly (Both) is the default mode. If you enter U, the mode used is the last mode set.

The syntax for this command is very flexible. If you enter U, the disassembly starts where the last U command left off. The G, J, R, and T series commands 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 18H. If you enter U <number> <length> the <number> is presumed to be an offset, and the segment is presumed to be CS. If a disassembly window is used, the size of the window overrides the length parameter.

If a disassembly window is used and it is active, the PgUp, PgDn, PadPlus, and PadMinus keys can be used to move forward and backward through memory. When using the up arrow or PgUp key to go backwards in the disassembly window, the value of IP will not go below zero. When disassembling memory without a disassembly window, the default length for disassembly is 18H bytes or lines, depending on the mode used.

---

		MAIN:	
1392:0000	55	PUSH	BP
1392:0001	8BEC	MOV	BP,SP
1392:0003	8B1000	MOV	AX,0010
1392:0006	9A6C029A13	CALL	CHKSTK
	FTOC#10:		
1392:000B	C746F60000	MOV	WORD PTR [BP-0A],0000
	FTOC#11:		
1392:0010	C746F22C01	MOV	WORD PTR [BP-0E],012C
	FTOC#12:		
1392:0015	C746F41400	MOV	WORD PTR [BP-0C],0014

---

*Figure 15-14. Disassembly of FTOC Program in UA Mode*

When UA mode is used (see Figure 15-14), all available symbols are displayed, including public and line symbols. (You can turn line symbol display off using the /L command.) The maximum backwards movement of the UA disassembly window is 21H lines. Any larger disassembly window will

---

not move back a full page.

Periscope supports disassembly of real and protected-mode opcodes for CPUs from the 8088 to the 80486. To set 16-bit or 32-bit disassembly, see the 16 and 32 commands. 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`.

Periscope evaluates register memory references of the current instruction, taking into account the current values of machine registers. For example, the instruction `MOV AX, [BP-10]` shows the symbol name `STEP` as a comment when the corresponding local symbol is used and is in the scope of the current procedure. Also, references such as `MOV [BX + SI-20],AX` are evaluated, showing any symbols found.

When `US` mode is used, the source and line numbers are shown, but no other symbols are displayed unless no source is available. (Source mode is the default mode as long as line number symbols are in the symbol table and `/E:0` is not used as an installation option.)

---

```
#7: {
    int lower, upper, step;
    float fahr, celsius;
#10:   lower = 0;           /* lower limit of temperature table */
#11:   upper = 300;        /* upper limit */
#12:   step = 20;          /* step size */
```

---

*Figure 15-15. Disassembly of FTOC Program in US Mode*

**WARNING:** What you see is not necessarily what you get when using source mode, since the source file may not match the code due to a source update or a code change. If in doubt, use `UB` mode instead of `US` mode. If the source file date/time stamp is more recent than the program's date/time stamp, however, the warning message, `source file more recent`



---

than program is displayed.

Use the **US** command to turn on source-level debugging (see Figure 15-15). This mode shows a minimum of assembly code, i.e., it shows assembly code until the first source line is found and then shows just the source code. You can use the up and down arrows and **PgUp** and **PgDn** keys to move through your source file. If you work in assembler, discover the joys of source-level debugging. It's incredibly helpful to be able to see your comments as you debug!

The source-line naming convention is composed of a module name of up to 8 characters, an asterisk, and the line number from 1 to 65535. For example, line 10 of **FTOC** is referred to as **FTOC#10**. While 'in' the module, the shorthand form **#10** may be used to reference line 10.

If a disassembly window is used, the line number of the first source line is always displayed for reference. If the line number is a valid symbol, it appears in the format **#nnnnn:**, where **nnnnn** is the line number. Otherwise, the reference line is shown as **(nnnnn)**. When the current instruction is a conditional jump, the **jump** or **no jump** comment is shown in the separator line of the disassembly window.

The module name is shown in the separator line if a disassembly window is used. If not, the module name is displayed as the first line of the disassembly. When possible, use a disassembly window so you can scroll through the source file.

If you're using a recent version of the linker, **Periscope** should never prompt you for a file name for source-level debugging. If you are ever prompted for a source file name, press **Alt-C** to see the name **Periscope** tried to use. Edit the file name as needed and press **return**. To fix the name permanently, see the description of the **MP** and the **MX** aliases in Chapter 10 (under **RS**). If you press **return** without entering a file name, source disassembly is disabled. To display the file name prompt again, enter **UA** and then enter **US**.

To display source code, the following conditions must be met:

- **DOS** must not be busy.

- A source file buffer must be available (if Periscope was installed with /E:0, the US command is not available).
- Line symbols must be available for Periscope to be able to associate an instruction with a source-code line.
- You must be disassembling memory at a source line symbol.

To test for these conditions, see the /L command.

---

```

                _MAIN:
#7: {
1392:0000 55          PUSH    BP
1392:0001 8BEC        MOV     BP,SP
1392:0003 8B1000      MOV     AX,0010
1392:0006 9A6C029A13  CALL    CHKSTK
#10: lower = 0;      /* Lower limit of temperature table */
1392:000B C746F60000      MOV     WORD PTR [BP-0A],0000
#11: upper = 300;    /* upper limit */
1392:0010 C746F22C01  MOV     WORD PTR [BP-0E],012C
#12: step = 20;      /* step size */
1392:0015 C746F41400      MOV     WORD PTR [BP-0C],0014

```

---

*Figure 15-16. Disassembly of FTOC Program in UB Mode*

When UB mode is used (see Figure 15-16), any available source code is displayed and line symbols are suppressed. This mode shows the source code, followed by assembly code generated by the source code.

### Examples:

U FTOC# disassembles memory in the current or default mode, starting at the first source line in FTOC. Use this technique to find the first source line in a module if you're not sure what the line number is.

U NEWPAGE disassembles memory in the current or default mode, starting at the symbol NEWPAGE. The default length of 18H bytes is used.

UA FTOC#10 L 1 disassembles memory in symbolic Assembly mode for one instruction starting at the symbol

---

**FTOC#10.**

**UB FTOC#10** disassembles memory in source-and-assembly (Both) mode starting at line 10 in FTOC.

**UB** disassembles memory in source-and-assembly (Both) mode starting where the last **U** command left off. If the **G**, **J**, **R** or **T** commands were used, the disassembly starts at **CS:IP**.

**US FTOC#10** disassembles memory in Source-only mode starting at line 10 in FTOC.

## **Command: View file (V)**

**Syntax:** **V** <file>

**Description:** This command is used to view a text file from within Periscope. Don't try to use it to display a file that's not in ASCII!

The <file> is any legal file name, including drive, path, file, and extension. To use this command, DOS must not be busy. A source file buffer must be available. If Periscope was installed with **/E:0**, this command is not available.

The file is displayed in the non-windowed area of the screen, unless a View window has been set up. The line numbers are displayed in the lower left-hand corner 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. Use the right arrow key to display beyond column 80 and the left arrow key or the **enter** key to get back. When you're finished viewing the file, press the **Esc** key to return to Periscope's prompt. If an EOF character (1AH) is found in the file, View treats it as the end of the file.

A simple string search is available. Enter a forward slash

---

and the text to be located. The search begins on the second line from the top of the screen. The search is not case sensitive. When found, the string is displayed at the top of the screen. To repeat a search, enter a forward slash and press Alt-D. If no match is found, the last line in the file is displayed.

To position to a specific line, enter a pound sign (#), the decimal line number (1 to 65535), and press return. The desired line is then displayed.

If a View window is used, the window is static after Esc has been pressed to return to the Periscope prompt. If the windows are changed or the screen is cleared, the View window is lost. Use the View command again to re-display the file.

**Example:**

**V C:PS.DEF** displays the file PS.DEF. Use the PgUp, PgDn, Up, Down, Left, Right, Home, and End keys to move through the file. When done, press Esc to return to the Periscope prompt.

## **Command: View Source file (VS)**

**Syntax: VS**

**Description:** This command is used to view the current source file. It has no arguments and is available only when source-level debugging has been turned on using the UB or US commands. This command functions exactly like the View command described above.

**Example:**

**VS** displays the current source file. When finished viewing the file, press Esc to return to the Periscope prompt.

---

## Command: Watch (W)

**Syntax:** `W*` or `W<byte> *` or `W[<byte>] [<format>] [<pointer>] <range>`

**Description:** This command is used to watch memory locations and I/O ports.

The Watch command is used in conjunction with a watch window, which may be up to 8 lines long. Each line shows the watch variable number, the format type in parentheses, the symbol name or address, and the desired value. The display of each watch variable is limited to one line on the screen. The formats available are the same as with the display command, plus P to watch an I/O port.

For example, to display the variable STEP in integer format, enter `W I STEP` or `W0 I STEP L2`. When no `<byte>` is specified, the next available value is used. To watch an I/O port, use `W P <port>`. To clear the entire watch window, use `W*`. To clear an individual watch variable, use `Wn *`, where n is from 0 to 7. To display all watch settings, use `W?`

The Watch command supports the use of pointers. To use this feature, enter a bracket ([]) or a brace {} after the display format and before the address. If the bracket/brace immediately follows the `<format>` (no space!), the pointer is dynamically evaluated each time the watch item is displayed.

If the bracket/brace does not immediately follow the `<format>` (one or more spaces), the `<pointer>` is evaluated when the watch command is entered and remains static after that time. For example, `W B{ FOO` dynamically evaluates the far pointer FOO, whereas `W B {FOO` is evaluated just at the time the watch command is entered.

### Examples:

`W0 I AMOUNT` displays the variable AMOUNT in integer format.

---

**W3** \* clears the watch variable number 3.

**W P 310** displays the value read from port 310H.

**W\*** clears all watch variables.

**W?** displays all watch settings.

## **Command: Write Absolute disk sectors (WA)**

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

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

The segment defaults to CS if no segment is specified in the <address>. The <drive> is either a single-digit number indicating the disk drive (0 = A, 1 = B, etc.) or A:, 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 not be busy. This command uses DOS interrupt 26H. See the DOS manual for information on the numbering of the absolute disk sectors.

**NOTE:** When using this command, be very careful. An absolute disk write can very easily destroy the file allocation table (FAT) or the 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 A: 10 20** writes data from memory starting at DS:100 to drive A:, starting at sector number 10H for 20H sectors.

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

## Command: Write Batch file (WB)

**Syntax:** WB <file>

**Description:** This command is used to write a 'batch' file that contains Periscope commands to save the current settings for the windows and the hardware and software breakpoints. To restore the state in a later debugging session, use the LB command to load the file. Unless an extension is specified, the default of .PSB is used.

### Example:

Enter **WB SAVE** to write the breakpoint and window settings to the file SAVE.PSB. Later, use **LB SAVE** to reload the breakpoint and window settings.

## Command: Write alias and record Definitions (WD)

**Syntax:** WD <file>

**Description:** This command is used to write the current alias definitions and record definitions to a PSD file. The <file> field is the name of the file to be written. An extension of .PSD is presumed. To use this command, DOS must not be busy.

---

**Example:**

**WD** **FTOC** writes the current alias and record definitions to the file **FTOC.PSD**.

## **Command: Write File to disk (WF)**

**Syntax:** **WF** [**<address>**]

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

The optional **<address>** specifies where the memory image of the file begins. If an address is not specified, **CS:100** is used. To use this command, **DOS** must not be busy. 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** is missing its header and is executable only at its current address.

**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: Write Symbols to disk (WS)

**Syntax:** WS <segment> <file>

**Description:** This command is used to write a Periscope symbol (PSS) file using the current symbol table.

The <segment> contains the relocation factor that is subtracted from the current symbol segment before the file is written. For COM files, this is the value of the PSP segment or CS. For EXE files, this is the value of the PSP segment plus 10H. The <file> is the path and file name of a PSS file. To use this command, DOS must not be busy.

This command cannot be used to write a MAP file. Only PSS files are supported. The symbol tables are left unchanged by this command.

If an error occurs when writing the symbol file, the symbols may be left with the relocation factor subtracted. If this happens, you can recover the symbols using WS 0 <file> to write the symbols without further relocation. Then use LS \* to clear the symbol table, followed by LS <segment> <file> to restore the symbol table.

### Examples:

WS CS SAMPLE subtracts the current value of CS from the symbol's segments and writes the file SAMPLE.

WS 0 C:TEST subtracts zero from the symbol's segments and writes the file C:TEST.PSS.

## Command: translate (X)

**Syntax:** X|XH <number> or XA <address> or XD <decimal number>

**Description:** X or XH is used to translate a one- to four-digit

---

hexadecimal <number> to its decimal, octal, binary, and ASCII equivalents. It may also be used to perform in-line arithmetic.

**XA <address>** 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.

**XD <decimal number>** is used to translate a one- to five-digit decimal number to its hexadecimal, octal, 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.

**Examples:**

**X 5051** displays:

5051h    20561d    050121o    0101 0000 0101 0001b    PQ

**XA 1234:5678** displays:

17988

**XD 20561** displays:

5051h    20561d    050121o    0101 0000 0101 0001b    PQ

## **Command: Use 16 or 32-bit disassembly (16/32)**

**Syntax:** 16 or 32

**Description:** 16 sets 16-bit disassembly. 32 sets 32-bit disassembly for use on 80386 or later CPUs.

There is currently no effective address support for 32-bit instructions.

---

**Examples:**

16 switches to 16-bit disassembly.

32 switches to 32-bit disassembly.

**Command: Option ditto (copy Periscope's screen)**

**Syntax:** /"

**Description:** This command is used to copy Periscope's screen onto the other display. It is usable only when a dual-monitor system that contains one color display and one monochrome display using an MDA is used.

**Example:**

/ " copies Periscope's screen to the other display.

**Command: Options 1 and 2 (switch symbol tables)**

**Syntax:** /1 or /2

**Description:** This command is used to switch to an alternate symbol table.

To use it, you must specify two symbol tables when Periscope is loaded, using two /T : xxx installation options. The /1 command makes the first symbol table active and the /2 command makes the second symbol table active.

You can use the LS command to manually load the current symbol table or use RUN to load the desired symbol table. RUN defaults to loading symbol table 1, but loads table 2 when RUN /2 <file> is used.

---

When debugging a program that uses overlays, the symbol table that contains overlays must be loaded last.

**Examples:**

`/1` makes the first symbol table active.

`/2` makes the second symbol table active, presuming that two `/T` installation options were specified when Periscope was loaded.

## **Command: Option A (toggle DOS Access)**

**Syntax:** `/A`

**Description:** This command is used to enable or disable DOS access by Periscope.

The `/A` command toggles Periscope's use of DOS. When used once, it turns Periscope's use of DOS off. When used again, it turns Periscope's use of DOS back on.

**Example:**

`/A` enables/disables DOS access by Periscope.

## **Command: Option C (display and set Colors)**

**Syntax:** `/C` [`<byte>`]

**Description:** This command is used to display and set the screen colors. The optional `<byte>` is the color attribute. If no number is entered, the colors for 00 to 7FH are displayed. If a number is entered, the screen color is set as in the `/C` installation option.

---

### Examples:

`/C` displays the available screen colors.

`/C 17` sets the Periscope screen color to white on blue.

## Command: Option D (Data window select)

**Syntax:** `/D [<byte>]`

**Description:** This command is used to select the active data window when more than one data window is in use.

The active window is the one modified by display commands. The inactive window(s) display memory in the same format as when they were last active. If only one data window is in use, this command has no effect. If no number is entered in the `<byte>` parameter, the next data window is made active (as indicated by the up arrow after the window number). If a number that corresponds to a data window (0-3) is entered, that window is made active.

### Examples:

`/D` makes the next data window active. If there are three data windows, the first use of this command makes the second window active. The second use of this command makes the third window active. The third use of this command makes the first window active, etc.

`/D 3` makes the last data window active, assuming four data windows are in use.

## Command: Option E (Echo screen to a file)

**Syntax:** `/E [<file>]`

---

**Description:** This command is used to echo Periscope's screen output to a <file>.

All non-windowed output is written to a disk file at the same time it is being written to the screen. To begin this mode, enter /E followed by a file name and a carriage return. While active, the command prompt shows /E to remind you that echo mode is on. To end echo mode, enter /E with no file name. The usual rules about DOS availability from within Periscope apply.

**Examples:**

/E D:OUTPUT starts echo mode, using the file D:OUTPUT. Until another /E command is used, Periscope's non-windowed screen output is written to this file.

/E ends echo mode, closing the file D:OUTPUT and returns to the standard Periscope prompt.

## **Command: Option K (capture Keystrokes to a file)**

**Syntax:** /K [<file>]

**Description:** This command is used to capture keystrokes to a <file>.

To start keystroke capture, enter /K <file>. Periscope then writes all keystrokes to the file until the /K command is used again. To replay the captured keystrokes, use the LB command. Note that the default file extension is .PSB. While this command is in use, Periscope's prompt becomes /K>. No keystrokes are captured while the menu system is active.

**Examples:**

/K TEST starts capturing keystrokes to the file TEST.PSB.

---

/K turns off keystroke capture and closes the file TEST.PSB.

## **Command: Option L (display Line symbols and source debug status)**

**Syntax:** /L

**Description:** This command is used to enable/disable the display of line-number symbols when the UA mode is used and to display the status of various items required for source-level debugging.

If no line symbols are found, the message `No lines found` is displayed. If you see this message, correct your compile and link options to get line numbers in the symbol file. They are required for source-level debugging.

If the source buffer size has been set to zero (`PS /E:0`), the message `No buffer found` is displayed. If you see this message, reinstall Periscope with a source file buffer, since it is required for source-level debugging.

If DOS is busy, the message `DOS busy` is displayed. Periscope uses DOS to access your source file, so DOS must not be busy for source-level debugging.

If no problems are found, the message `lines found` is displayed.

**Example:**

/L toggles the display of line-number symbols when the UA mode is used. It also displays the messages described above.

---

## Command: Option M (Messages)

**Syntax:** /M <byte>

**Description:** This command is used to control the types of messages displayed when debugging Microsoft Windows applications. The <byte> defaults to 0 (no messages), but these values can be entered: 1 for allocation messages only, 2 for movement messages only, or 3 for allocation and movement messages.

**Example:**

/M 1 causes allocation messages only to be displayed when debugging Microsoft Windows applications.

## Command: Option N (Nearest symbols)

**Syntax:** /N [<address>]

**Description:** This command is used to search for the symbols nearest to the specified <address>.

Up to three symbols are displayed on up to three lines: the next lower symbol (preceded by >), the equal symbol (preceded by =), and the next higher symbol (preceded by <).

This command can help you get your bearings when you've interrupted an executing program by showing you the nearest symbols. If no <address> is entered, CS:IP is assumed. The nearest symbol that is located lower in memory is displayed on the first line, the symbol for the specified address is displayed on the next line, and the nearest symbol that is located higher in memory is displayed on the next line. If no lower, equal, or higher symbol is found, nothing is displayed.



---

### Examples:

Assume three symbols X, Y, and Z located at 1000:100, 1000:200, and 1000:300 respectively.

`/N 1000:200` displays:

```
> X
= Y
< Z
```

`/N 1000:0` displays:

```
<X
```

## Command: Option Q (Quiet)

**Syntax:** `/Q`

**Description:** This command is used to turn off Periscope's display output until the end of the current command line.

This is useful when you want to suppress Periscope's output to the screen. For example, when using the `/E` command to echo Periscope's output to a file, you can suppress the display using this command.

### Example:

`/Q;U IP L1000` turns off the display output until the disassembly is complete.

## Command: Option R (Remove symbol)

**Syntax:** `/R <symbol>`

---

**Description:** This command is used to remove public and line symbols (but not local symbols) from the symbol table. For instance, since symbol names are evaluated before register names, a symbol named AX would disable references to register AX unless this command was used to remove the symbol.

**Examples:**

`/R AX` removes the symbol AX, leaving no conflict with the register named AX.

`/R FTOC#10` removes the symbol FTOC#10. Be careful when removing line-number symbols, since these cannot be re-entered using the ES command.

## **Command: Option S (Segment change)**

**Syntax:** `/S <segment> <segment>`

**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, including local data 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. This command is used to adjust the segments of symbols when a program relocates its code or data areas.

**Example:**

`/S 1234 DS` changes the segment of all symbol table entries that are currently 1234 to the current value of DS.

---

## Command: Option T (Trace interrupt table)

**Syntax:** /T [?] [\*] [#] [<byte>] [...]

**Description:** This command is used to force tracing of interrupts when the GT command is used.

Some interrupt service routines turn the trap flag off when returning status information in the flag registers. If Periscope does not trace all the way through such routines when the GT command is used, the program can get out of Periscope's control and begin executing at full speed. The known troublesome interrupts are 13H, 15H, 16H, 1AH, 20H, 25H, 26H, 2FH, 40H, and 41H. When Periscope is first installed, these interrupts are flagged for forced tracing. Using this command, you can change the interrupts that are to be traced when GT is used. The possible command arguments are \*, #, ?, and numbers from 0 to FF (always presumed hex). The \* clears all traps and # sets all traps. A ? displays the current trace list. A hex number toggles the state for that interrupt from off to on or vice-versa.

### Warnings:

- Interrupt 21H should be in the trace list whenever function 4BH (Exec) is used or if Borland's SIDEKICK is in the system.
- When an interrupt is not traced, Periscope becomes dormant until the second instruction after the INT XX instruction.
- If a GT command is used when the current instruction (CS:IP) is an interrupt, the interrupt is always traced.
- If you have problems with the GT command losing control, try using the GA command.

### Examples:

/T # forces tracing of all interrupts.

/T \* 21 clears all interrupts and then forces tracing of Int 21H.

---

## Command: Option U (User exit)

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

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

To use this command, a program similar to USEREXIT.ASM must be installed and Periscope must be installed with the /I option. The <byte> entered after the /U command must be from 9 to FFH. It cannot be a literal. The number is passed to the user-written program in register AH. Other information is passed, including the optional <address> entered on the command line. This parameter may be followed by additional free-form information. This means you cannot stack commands on the line following a /U command. When a user exit command is entered, Periscope displays an error if the signature of the user exit code cannot be found.

The user exits from F0 to FFH are reserved for Periscope. See the USEREXIT.ASM program for more information.

USEREXIT has a status display for the numeric processor. To use USEREXIT, load it before Periscope is loaded. Then, use /I : 60 when installing Periscope. 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 Periscope had the /I : 60 installation option, /U 9 executes user exit number 9.

## Command: Option W (Window setup)

**Syntax:** /W [<token>] [[:<byte>] [.<color>]] [...], where <token> is D, R, S, U, V, or W; <byte> is the length in lines; and <color> is the color

---

attribute; or /W -<token>; or /W +<token>; or /W ?

**Description:** This command is used to change Periscope's windows. Its use and syntax are identical to the /W installation option.

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

The <token> can be D, R, S, U, V, and/or W to 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, except for the stack and vertical register windows, which are always on the right-hand side of the screen.

The <byte> parameter defines the length (number of lines) of the horizontal window in hex (except for the R window). If no length is specified, a default is used. The maximum length for any one window and the total area that can be windowed is four lines less than the screen length, including a separator line following each window. When a length specification is used, at least one space must follow the number. If you're using 43-line mode or 50-line mode, the windows can get quite large. Since the windows are regenerated after each command, large windows can slow Periscope's response time. The default and minimum number of lines for each of the horizontal window types (vertical window lengths are determined by the total number of horizontally-windowed lines) are shown in the table in Figure 15-17.

The colors of the windows can be individually set, using a hex number from 1 to 7F in the <color> parameter. The numbering scheme is the same as that used by the /C installation option, and by the /C command. To set a data window of five lines using color 1FH, /W D:5.1F would be used.

---

	DEFAULT	MINIMUM
Data	4	1
Register	2	2
Unasm	4	1
View	4	1
Watch	4	1

---

*Figure 15-17. Periscope Window Lengths*

**Data Window.** The **D** token sets up a data window to show data in any of the display formats. 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. When **RUN** is used to enter Periscope, the display address is set to **DS:100**.

Up to four data windows may be used, with each window showing a different range and using a different display format. For example, if you want to set up three data windows with lengths of 4 lines, 2 lines, and 6 lines, respectively, enter **/W D:4 D:2 D:6**. To change the active data window, use the **/D** command. If the start address of a data window matches a symbol, the symbol name is displayed in the separator line at the end of the window. When the data window is active, the **PgDn**, **PgUp**, **PadPlus**, and **PadMinus** keys may be used to move forward and backward through memory.

**Register Window.** The **R** and **R:2** tokens set up a horizontal register window to show register and flag information. The length is fixed at two lines. The effective address of any memory reads or writes is shown in the separator line following this window.

The **R:1** and **R:3** tokens establish a vertical window on the right-hand side of the screen to show standard and 80386 register displays, respectively. The displays require 18 lines of window space to display the full register set and the flags. If fewer lines are available, the register displays are truncated. A vertical register window cannot exist without other windows. These vertical windows may be toggled off and on

---

using the Alt-R (standard) and Alt-3 (80386) keys. When either of these windows are used, the effective address is displayed in the separator line following the Watch window.

**Stack Window.** The S token sets up a vertical window on the right-hand side of the screen to show the stack. The length of the stack window is equal to the total number of lines contained in the other types of windows. A stack window cannot exist without other windows. A chevron in the left margin of the window indicates the current value of the BP register. The stack window may be toggled off and on using the Alt-S keys. The stack is read from the upper right-hand corner of the screen downwards. If you choose not to use a stack window, you can always view the stack using DW SS:SP.

**Disassembly Window.** The U token sets up a disassembly window to show disassembled instructions. 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-series command is used. Any area of memory can be disassembled by using the U command with the desired address. Sticky code breakpoints are highlighted when shown in the disassembly window. The current instruction is shown in reverse colors. (Some window colors may cause the reverse color to be invisible. For example, color 7E on a monochrome EGA causes the reverse bar to be invisible.)

The maximum backwards movement of a disassembly window is 21H lines. Any larger disassembly windows will not move back a full page. If you 'lose' the reverse video bounce bar, use the R command to redisplay it. When the disassembly window is active, the PgDn, PgUp, PadPlus, and PadMinus keys may be used to move forward and backward through memory.

In the separator line following the disassembly window, the current location is shown. Periscope uses the DOS memory allocation blocks to get the actual program name when possible. Two caveats: the lookup is based on CS only and In RUN.COM will be shown when in a program loaded by RUN unless RUN /T or RUN /X was used.

**View Window.** The V token is used to establish a View win-

---

dow to display a text file. The View command uses the space reserved by the window. See the description of the View command for more information.

**Watch Window.** The **w** parameter sets up a Watch window to display the contents of up to eight memory locations or I/O ports. Once a Watch window is established, the Watch command is used to specify the locations to be "watched". When the Watch window is active, the PadPlus and Pad-Minus keys may be used to scroll through the window. See the Watch command for more information.

Ctrl-F9 is used to restore the original window settings used when Periscope was installed. Ctrl-F10 is used to restore the most recent window settings.

The **/W -<token>** and **/W +<token>** commands are used to decrease or increase the length of a window by one line. They will not create or remove a window. To decrease the length of the data window, use **/W -D**. To increase the length of the watch window, use **/W +W**. When decreasing a window's length, the minimum resulting length is one line. When increasing a window's length, the maximum resulting length is determined by the maximum length for the screen.

The **/W ?** command displays the current window settings.

#### **Examples:**

**/W D:8.1F R** windows data in the first 8 lines of the screen using color 1F (white on blue) followed by two lines of register information. A total of 12 lines are used for windows, including the two separator lines.

**/W SRU** windows register information in the first two lines of the screen, followed by four lines of disassembly. A total of 11 lines are used for windows, including separator lines, so the specified stack window is 11 lines long.



---

## Command: Option X (eXit to DOS)

**Syntax:** /X

**Description:** Exit to DOS, presuming DOS is not busy and memory has been freed.

DOS must not be busy and memory must have been freed using DOS function call 4AH. To return to Periscope, enter **EXIT** at the DOS prompt. While at the DOS prompt, don't execute **RUN**, **PS**, or change the state of the system (e.g. change monitors).

**Example:**

/X exits Periscope and displays the DOS prompt, assuming memory has been freed and that DOS is not busy. When you're ready to return to Periscope, enter **EXIT** at the DOS prompt. ♦



15

---

# Messages



- **Informational Messages and Prompts**
- **Error Messages**

**Y**ou'll find details on informational messages, prompts, warnings, and error messages displayed by Periscope in this chapter.

---

## INFORMATIONAL MESSAGES AND PROMPTS

The following messages and prompts are generated by programs in the Periscope package and are listed in alphabetical order:

### **\*Break\***

This message is displayed when the break-out switch is pressed or an NMI is generated while Periscope is already active.

### **Breakpoint cleared**

This message is displayed when a breakpoint is re-entered.

### **DOS 3.00 or later required**

Version 4.32 is the last version of Periscope that runs under DOS 2.x.

### **EOI issued for IRQ x**

This message is displayed when Periscope must issue an End of Interrupt for a hardware interrupt. The IRQ level designated by **x** indicates the interrupted activity: 0 = timer (INT 8); 1 = keyboard (INT 9); etc.

### **Exception Int x**

On an 80286 or later CPU, Periscope can intercept two types of exceptions, illegal opcode (INT 6) and segment wraparound (INT 0DH). The **x** is either 6 or D.

### **Grrr!**

This message is displayed when the watchdog timer on an IBM PS/2 machine activates Periscope after detecting a hung system.

### **Parity error 1**

This message is displayed when a motherboard parity error occurs. Make sure you've entered the correct response to the Periscope configuration option that asks if you have a PC/XT motherboard with an 80286 or later turbo card. If your configuration is correct, run your system diagnostics.

### **Parity error 2**

---

This message is displayed when a parity error occurs in the expansion bus or when the Periscope II break-out switch is used.

**NOTE:** This is normal when the Periscope II break-out switch is pressed. It can also occur when two Periscope boards are used in the system. Try plugging the break-out switch into the other board.

**Press Esc to end full-screen mode**

This message is displayed when Periscope is in full-screen mode and an unexpected keystroke is entered.

**Source buffer too small**

This message is displayed when the Periscope source file buffer is less than 1/32nd the size of the source file in use. Increase the size of the buffer using the /E installation option for better performance.

**Source file?**

This prompt is displayed when Periscope is unable to find the source file. Press Alt-C to see the name Periscope used. Correct the name and press return. Use the MP and MX aliases as needed to automate the source file access.

**Source file more recent than program**

This message is displayed when the date/time stamp on the source file is later than the date/time of the program file. An incorrect source line may be displayed if the two files do not match.

**Warning: Timer interrupts (IRQ 0) are turned off via port 21H**

Periscope has found that the system clock has been turned off. This is not a normal condition, so we are alerting you to it.

**Warning: Keyboard interrupts (IRQ 1) are turned off via port 21H**

Periscope has found that the keyboard has been turned off. This is not a normal condition, so we are alerting you to it.

---

## ERROR MESSAGES

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

- 01 through 39 – resident portion of PS.COM
- 40 through 89 – transient/installation portion of PS.COM
- 90 through 99 – RS.COM
- 100 through 129 – TS.COM
- 130 through 139 – SYMLOAD.COM
- 140 through 149 – INT.COM
- 150 through 159 – PSKEY.COM
- 160 through 169 – PSTEST.COM
- 170 through 189 – RUN.COM
- 190 through 199 – SYSLOAD.SYS
- 200 through 229 – PS3TEST.COM
- 230 through 239 – CONFIG.COM
- 240 through 249 – SETUP.COM
- 270 through 279 – PSTERM.COM
- 400 through 413 – PS4TEST.EXE

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

### 01 – Invalid command

An unknown Periscope command was entered. Enter ? to display the commands available.

### 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 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.

### 03 – Missing segment

Some commands that modify memory 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.

**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 commands require a byte/string list.

**07 – Missing quote**

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

**08 – Invalid/missing operator**

An expected argument was not found. Check the command syntax and try again.

**09 – Number is not decimal**

A decimal number must be composed of the digits 0-9 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`, `FS`, `GS`, `SS`, `CS`, `IP` or `FL`. 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 breakpoints are set for the command. See Chapter 15 for the limits for the command used.

**14 – Wrong version in PSS file**

The version number found in the PSS file is not current. Regenerate the PSS file using TS.

**15 – Can't trace INT 3**

An attempt was made to trace interrupt 3 using Periscope. This interrupt is off-limits, since it points to Periscope and any attempts to have Periscope trace itself would result in total confusion.

**16 – Can't 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 – 2nd address/port < 1st**

The second address or port number is 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 may be preceded by a period and must be followed by a delimiter such as a space, carriage return, or semi-colon. The maxi-



---

mum symbol length is 32 characters. To display the symbol names and addresses from the symbol table, use Alt-I. To display the record definition table, use Alt-E.

#### **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 symbol table is full, reload Periscope with a larger /T installation option. If the record definition table is full, reload Periscope with a larger /R installation option.

If the table is invalid, chances are good that it has been garbled. For the symbol table, use the LS command to clear and reload symbols. For the record definition table, use the LD command to clear and reload definitions.

#### **20 – DOS busy**

DOS function calls are used by the Load, Name, View, Unassemble Source, Write, Echo and DOS eXit commands. Since DOS is not re-entrant, Periscope tests to be sure that DOS is available (not busy). This is done by checking a flag set by DOS. This flag must be zero and interrupts must be enabled for Periscope to allow DOS functions. If you receive this message, use the Go command to get back to your program's code or go to the next invocation of INT 28H, using G {0:28\*4}. Then try the command again.

#### **21 – Not enough memory**

Insufficient memory is available to perform the Load or eXit to DOS command. When using the /X command, you must first release memory back to DOS.

#### **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, respectively, had an invalid drive identifier.

#### **23 – Can't 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 – Invalid 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, U, V and W 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 the number of lines desired. For the R token, the number may be 1, 2, or 3. A number must be followed by a space, a slash (indicating the start of another installation option), a carriage return, or a color setting in the format .cc, where cc is a number from 01 to FF. The total number of windowed lines, including a separator line for each window, must be 21 or less. If 43-line or 50-line mode is used, the window sizes may be larger by the corresponding number of 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. Also check to see if DOS has become busy while the /E or /K command is in us.

#### **26 – Function not available**

This error indicates that the desired command is not available. It can occur under various conditions:

- when an IC or IR command is used and no IS command has been previously used to save the interrupt vectors
- when an RC or RR command is used and no RS command has been previously used to save the registers
- when a TB, TR, or TU command is used and /B:0 was used when Periscope was installed
- when a UB, US or V command is used and /E:0 was used when Periscope was installed
- when LS is used and /T:0 was used when Periscope was installed
- when a BU or /U command is used and no /I:nn was used when Periscope was installed or the user exit has

- 
- been corrupted
  - when LD is used and /R:0 was used when Periscope was installed
  - when a QR command is used and DOS is busy or RUN was not used to enter Periscope
  - when a /2 command is used and no second symbol table is present
  - when the /R command is used to remove a local symbol
  - when the BD command is used on a CPU prior to the 80386

#### **27 – Unknown mnemonic**

An unknown mnemonic was specified to the in-line assembler. The assembler knows the mnemonics for the 8086, 8087, 8088, 80186, 80286, and 80287 processors. For the 80286, only the real-mode opcodes are supported. Check the mnemonic and try again. Prefixes other than segment overrides must be on a separate line preceding the instruction they affect.

#### **28 B, W, D, Q or T pointer needed**

An ambiguous instruction was specified to the in-line assembler. Some 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. 8087/80287 instructions may require a width indicator of D, Q, or T for double word, quad word, or ten byte respectively.

#### **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 – Invalid 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 – File too large**

The size of the file being loaded is too large to fit in the first 640K of memory. Use a lower load address if possible.

**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.

**35 – COMSPEC not found**

The `/X` command was unable to find the COMSPEC parameter in the environment block. DOS may be garbled. Reboot and try again.

**37 – Range conflicts with PS**

The range specified by an `HM` or an `HP` command crosses into Periscope's memory or ports. Specify another range that does not interfere with Periscope.

**38 – Internal error**

An error has occurred in Periscope's paged memory allocation or in Model IV's hardware breakpoint settings. Please call Tech Support if you get this error.

**39 – Invalid HM/HP settings**

The current and/or true states used with the `HM` or `HP` commands are invalid. The state numbers may be from 0 to 6 (when `HC TT` is used, the upper limit is 5). Also, any true state must be used as a current state.

**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`, `/K`, `/R`, `/S`, 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. To display the valid installation options, enter **PS ?** from the DOS prompt, or use **PS \*** to install Periscope.

**43 – Interrupt must be 08H, 09H, 10H, 15H, 16H, 17H, or 1CH**

The **/V** option specified an interrupt number other than the ones listed above.

**44 – Error using protected memory -- run PSTEST**

Periscope was not able to properly install itself in the protected memory. Check the port setting on the board and the memory and ports specified with the **/M** and **/P** installation options, if any. If the problem persists, run **PSTEST** (see Chapter 10).

**45 – Error reading PS1 .COM-- run CONFIG**

The transient portion of Periscope (PS) was not able to load the resident portion (PS1). Be sure that both PS and PS1 are in the same directory. If you can't find PS1, you probably haven't run **CONFIG**. When Model III or IV is used, either PS3 or PS4 is also needed and must be in the same directory.

**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. If the problem persists, run **PSTEST** (see Chapter 10).

**47 – Screen size must be from 0 to 40H (64) KB**

The size of the program's screen specified with the **/S** option must be from zero to 40H K. Note that the number is in hex!

**48 – Symbol table size must be from 0 to 1FFH (511) KB**

The size of the symbol table specified with the **/T** option must be from zero to 1FFH K. Note that the number is in hex!

**49 – INT 2 does not point to Periscope -- check for conflicts!**

---

The NMI vector is not pointing to Periscope. Check to make sure that your display adapter is not running in an emulation mode where it is using NMI. See Appendix B for more information.

**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 – Unable to remove Periscope from memory**

Periscope was unable to release memory. Use PMAP or MAPMEM or a similar program to check the contents of memory.

**52 – Unable to read Help or Comment file**

An error occurred reading PSHELP.TXT or PSINT.DAT. Rerun CONFIG to regenerate these files.

**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 or /W installation 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/write response file**

An error occurred when Periscope tried to read or write the response file. Check the file name or disk 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 – Trace buffer size must be from 0 to 3FH (63) KB**

The size of the software trace buffer specified with the `/B` option must be from zero to 3FH 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 2000H paragraphs. If the PSP is C00H and the top of memory is A000H, then the allowable range for the load segment is C10H through 8000H.

**61 – Source buffer size must be from 0 to 8KB**

The size of the source buffer specified with the `/E` option must be from zero to 8K.

**62 – IRQ masks must be from 0 to FFH**

The values specified for the IRQ mask must be entered as one byte from 0 to FFH or two bytes from 0 to FFH, separated by a space.

**63 – Periscope not correctly configured -- run CONFIG**

Periscope must be configured before it can be run in your system. See Chapter 4.

**64 – Unable to use 43- or 50-line mode**

An EGA is required for 43-line mode. A VGA is required for 50-line mode.

---

**65 – Defective CPU (stack change was interrupted). Replace ASAP!**

The 8088 CPU in your system is an early version of the chip that does not protect the instruction after the stack segment is modified. This defect can cause problems when tracing through DOS, using a numeric processor, and when using Periscope. The CPU should be replaced as soon as possible.

**66 – Incorrect software for Periscope hardware or CPU type**

An attempt was made to run the Periscope software when it is configured for a board that is not in the system or for a board that is not supported on this CPU.

**67 – Unable to read trace buffer -- run PSxTEST**

Periscope was unable to read the hardware trace buffer. Check the items listed under the description of the appropriate diagnostics program PS3TEST or PS4TEST (Chapter 10) and try again.

This error may also show a message of the form `Error segment: xxxx, Error code y`, where `y` may be one of the following:

- 0 - start state for confidence test
- 1 - more than 32 entries found in confidence test
- 2 - zero entries found in confidence test
- 3 - matching record is not a word or dword
- 4 - no matching records found in confidence test

**68 – Unable to initialize COM port**

An error occurred when Periscope attempted to initialize the COM port for alternate PC or remote support. Check the port number and speed settings on both sides and try again. Also confirm that your null-modem cable completely matches the specifications given in the file NOTES.TXT.

**69 – Error reading PMx.COM--run SETUP**

The menu program file PMx.COM was not found. Re-run SETUP (see Chapter 4).

**70 – Interrupt 1CH does not point to an IRET instruction**

Periscope's dynamic configuration was not able to find an IRET instruction to correlate INT 1CH to. If you encounter



---

this error, please call Tech Support.

**71 – Invalid com port number (1-8) or speed (S, M, or F)**

When an alternate PC or remote debugging is used, the com port number may be specified as /AV:px, /AK:ps, or /2:px, where p is the port number and x is the speed (Slow, Medium, or Fast). Both systems must be using the same speed.

**72 – Installation option invalid when remote debugging is used**

When remote debugging is used, you cannot use the /AK or /AV installation options. Also, Periscope III cannot be used in remote mode.

**73 – /M or /P installation option invalid for PS/2 system**

When Periscope I/MC is used, the memory or ports used by the board are read from the board and the /M and /P installation options are ignored.

**74 – /Q installation option invalid when /AK or /AV used**

When an alternate PC is used, the /Q installation option is not allowed. This is due to the fact that the alternate PC support requires a com port that will not be initialized at ROM-scan time.

**75 – Error reading xxKEYS.PSD**

This error is displayed when Periscope is unable to read the file xxKEYS.PSD, where xx is PS, CV, or TD for Periscope, CodeView, or Turbo Debugger function key usage, respectively. If the file exists, but is not in the current directory, use the DOS Set command (SET PS = C:\XXXX) to set the Periscope path. If the file does not exist, rerun SETUP from the original Periscope disk or a backup copy.

The xxKEYS.PSD loaded in the read-only area must be 511 bytes or less. If it is too large, modify the .DEF file used to generate it.

**90 – DEF file not found**

RS was not able to find a file of the specified name with an extension of DEF.

**91 – Unable to read DEF file**

---

An error occurred reading the DEF file. Check the file name 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 in Chapter 10.

**93 – Not enough memory**

Insufficient memory is available for RS to load the DEF file. Check the amount of available memory using CHKDSK and re-boot as needed.

**94 – Unable to write PSD file**

An error occurred when RS attempted to write the PSD file. Check the file name and try again.

**100 – xxx file not found**

TS was not able to find a file of the specified name with an extension of MAP. If the options so indicate, a SYM or EXE file is used instead of a MAP file.

**101 – Unable to read xxx file**

An error occurred reading the MAP, SYM, or EXE file. Regenerate the file and try again.

**102 – Line xxxxx of xxx file is not in correct format**

The MAP, SYM, or EXE 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. If this error occurs on an unmodified MAP file, please call Tech Support.

**103 – Not enough memory**

Insufficient memory is available for TS to load the MAP file. Check the amount of available memory using CHKDSK and re-boot as needed.

**104 – Unable to write PSS file**

---

An error occurred when TS attempted to write the PSS file.  
Check the file name and try again.

**105 – Unknown EXE symbol type**

TS encountered an unknown symbol type in the EXE file.  
Please call Tech Support if you get this error.

**106 – Unknown PLINK symtable type - x**

TS encountered an unknown record type in the symbol table at the end of the PLINK file. Please call Tech Support if you get this error.

**107 – Symbol table overflow at line xxxxx**

The compressed symbol table is too large. Use a text editor to modify the MAP file. You can either delete lines from the MAP file or comment them using braces, where { begins commenting and } ends it.

**108 – Invalid option**

An invalid command-line option was used. Enter TS ? to display the valid options.

**109 – EXE file contains inconsistent symbol information**

This error occurs when line numbers from two or more modules are competing for the same address space. Try changing any tricky segment usage or turning off symbol generation for any assembly routines in included files.

**110 – Line numbers out of sequence at line xxxxx**

Use the /LA or /LD options to accept or discard out-of-sequence line numbers.

**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.

**140 – File not found**

INT was unable to read the specified file. Check the file name and try again.

**141 – Unable to read or write file**

An error occurred reading or writing the indicated file.



---

Check the disk and try again.

**142 – Invalid option**

An invalid command-line option was used. Enter `INT ?` to display the valid options.

**150 – INT x does not point to Periscope!**

PSKEY cannot invoke Periscope using the indicated interrupt. If Model II-X is used, INT 3 must be used by PSKEY to activate Periscope. For all models, the indicated interrupt must point to Periscope. Use INT to display the current vectors. If an interrupt has been corrupted, use RUN to refresh them and try again.

**160 – Invalid option**

An invalid command-line option was used. Enter `PSTEST ?` to display the valid options.

**161 – xx test failed**

The indicated memory test failed. If widespread errors occur, check the memory and port settings for conflicts. If just a few errors occur, ensure that the memory chips are fully seated.

**162 – Page test x failed at page xx**

Test 1 checks to make sure that 512KB or 1024KB of paged memory is found. Test 2 writes the page number to each of the 8K pages. If either of these tests fail, check the memory and port settings used by the board. Be sure to use the appropriate `/M` and `/P` options as needed.

**163 – Rotate failed**

PSTEST was unable to write a rotating bit pattern to the protected memory. If just a few errors occur, ensure that the memory chips are fully seated.

**164 – Write protect failed**

PSTEST was unable to write protect the memory. Check the memory and port switch settings.

**165 – Copy test failed**

PSTEST was unable to copy a block of memory to the protected memory. If just a few errors occur, ensure that

---

the memory chips are fully seated.

**166 – Not enough memory**

Insufficient memory is available for PSTEST to run. Check the amount of available memory using CHKDSK and re-boot as needed.

**167 – Refresh test failed**

Data read from the board did not match the values previously written. Check the memory and port settings, and also check the technical reference for your system to be sure that it provides RAM refresh signals for memory cards. If errors persist, please call Tech Support.

**168 – Board running faster than rated speed**

If the board is in an 8MHz IBM AT, it is running in a borderline condition. To be safe, you may add a wait state using jumper J2. This error may also occur on busses running unusually fast.

**170 – INT 2 does not point to Periscope -- check for conflicts!**

See the explanation of Error 49 above.

**171 – 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.

**172 – Unable to read file**

An error occurred reading the file. Check to be sure the disk is ready and that the file size shown by DIR indicates the true file size.

**173 – 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.

**174 – 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.



---

**175 – Periscope not installed correctly**

RUN was unable to modify the protected memory. Reload Periscope and try again.

**176 – Internal error**

See the explanation of Error 38 above.

**177 – 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.

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 Alt-E to see the records that were loaded. Note that the last record will usually be only partially defined.

**178 – PSS file larger than symbol table - no symbols loaded**

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, or use the /T or /X option with RUN. Otherwise check the file and try again.

**179 – Symbol table full or invalid**

A logical error was found in the symbol table during final processing. Regenerate the PSS file and try again.

**180 – PSS file date/time is prior to program's date/time**

This warning message indicates that the date/time stamp on the program file is more than one minute later than that of the PSS file. This is to be expected if the program file has been patched. Otherwise, it indicates that an obsolete PSS file is being used. Proceed with caution!

**181 – Unable to execute TS**

RUN was unable to EXEC TS.COM. Manually run TS and then re-execute RUN.

**182 – Exec failure**

---

An error occurred when using the DOS EXEC function to load your program. Reboot and try again. If the problem persists, call Tech Support.

**183 – Invalid Exec symbol size**

The size indicated with the /x: option must be from 0-1FFH KB.

**184 – PSD file larger than record table - no records/aliases loaded**

An error occurred when RUN attempted to load the DEF or PSD file into the record definition table. Usually the DEF/PSD file is larger than the space reserved for the record definition table when Periscope was installed. If this is the case, restart Periscope with a larger record definition table. Otherwise check the file and try again.

**185 – File not found**

The program to be loaded by RUN was not found. Check the file name and try again.

**186 – Wrong version number in PSS file**

See the description of Error 14 above.

**187 – Remote timeout**

An error occurred when RUN tried to establish communications with the remote PC. Check to make sure that the null-modem cable is securely connected on both ends and that the target system is still functioning.

**188 – Only one symbol table found**

The /2 option requires two symbol tables. When PS.COM is loaded, be sure that two /T installation options are specified. For example, PS /T:20/T:40 would set up two tables.

**190 – Program terminated without resident request**

The program specified by the /P option did not terminate and stay resident. This is a warning message that can be suppressed with the /Q option.

**191 – Unable to read file**

The file specified with the /P option cannot be read.

---

**192 – Invalid option**

An invalid command-line option was found in SYSLOAD.SYS. Check the options used with the description of SYSLOAD.SYS in Chapter 10.

**200 – Invalid option**

An invalid command-line option was used. Enter `PS3TEST ?` to display the valid options.

**201 – Unable to write PSBUF.DAT**

An error occurred writing the hardware trace buffer to disk. Check the disk and command-line options used and try again.

**202 through 225**

These errors indicate a potentially serious diagnostic failure. Check the items listed under the description of the utility program PS3TEST in Chapter 10 and try again. If errors persist, call Tech Support.

**226 – Unable to read diskette**

PS3TEST was unable to read the saved trace buffer from a floppy disk. Check the disk and try again.

**230 – Unable to read PS2.COM**

CONFIG was unable to read PS2. Check the disk and try again.

**231 – Interrupt 1CH does not point to an IRET instruction**

Interrupt 1CH does not point to an IRET instruction. Check to make sure that no device drivers or memory-resident programs are installed and reboot the system as needed. If this does not clear up the problem, please call Tech Support for assistance.

**232 – Unable to read file**

An error occurred when CONFIG attempted to read a file. Make sure the current directory is a complete copy of the original distribution disk and try again.

**233 – Wrong model or version of Periscope**

The version or model number found in PS2 does not equal



---

the expected version or model. Restore the files from your backup disk and try again.

**234 – Unable to write file**

An error occurred when CONFIG attempted to write a file on the target disk. Check the disk and try again.

**235 – File too large**

An internal error has occurred. Reboot and try again. If the problem persists, please call Tech Support.

**236 – Format error in PSINT.TXT at line xxxxx**

The interrupt comment file is not in the correct format. Restore the file from the original and try again.

**240 – File PERI.PGM not found**

Please put the Periscope diskette into drive A and type  
**A:SETUP.**

**241 – Invalid drive or directory specified. Please try again.**

The target drive or directory is not available. Please check the name and try again.

**242 – Not enough memory**

Insufficient memory is available for SETUP to start the self-extract process. Check the amount of available memory using CHKDSK and re-boot as needed.

**243 – Unexpected error during self-extract process**

This usually indicates that the target disk is full. Check the available disk space and try again.

**244 – Disk error**

A disk error has occurred during the self-extract process. Check the target drive and try again.

**270 – xxxx timeout (Function = yy, byte = zz)**

This indicates a read or write timeout when PSTERM tried to read or write a byte across the serial link to Periscope. Retry several times, then if errors persist, select the Fail option. If things do not clear up, select the Abort option and restart PSTERM.COM and PS.COM.

---

#### **400 through 408**

These errors indicate a potentially serious diagnostic failure. Check the items listed under the description of PS4TEST in Chapter 10 and try again. If errors persist, call Tech Support.

#### **409 – Invalid option**

An invalid command-line option was used. Enter **PS4TEST ?** to display the valid options.

#### **410 – Unable to write PSBUF.DAT**

See the description of error 201 above.

#### **411 – Unable to read diskette**

See the description of error 226 above.

#### **412 – Cannot run in Virtual 86 mode – remove 386 control program**

PS4TEST must be run in real mode. If you have a 386 control program such as 386MAX, QEMM, CEMM, etc., remove it while running PS4TEST. You may have these types of programs running while Periscope is active, but PS4TEST requires a clear environment for accurate testing of the Model IV board.

#### **413 – Break-out switch failed**

The test of the break-out switch failed. Check the items listed under the description of PS4TEST and try again. If errors persist, call Tech Support. ♦

# Tech Support and Trouble- shooting

- Tech Support
- Troubleshooting

**T**ech support procedures and the most common problems are discussed in this appendix. Please check to see if your problem is discussed before calling Tech Support. Also, please do either call or contact us via our BIX vendor support conference rather than writing or FAXing if at all possible. It's much faster and easier to resolve problems on the telephone or BIX than by mail or FAX.



---

## TECH SUPPORT

If you are a registered Periscope user, you may receive free Tech Support by calling 404/875-4726, Monday through Friday, from 1PM to 5PM Eastern time. We also run a vendor conference (Periscope) on BIX, so if calling is not possible or convenient, you may use BIX as a forum for your questions and/or problems. We will return Tech Support calls to registered users in the U.S. and Canada when we are not available during Tech Support hours.

When you call with a problem, please have your registration number handy. If you purchased your Periscope recently from The Periscope Company (TPC) but have not sent your registration card in, have both your invoice number and your registration number available. If you did not purchase your Periscope from TPC, we will assist you in the installation and configuration of your Periscope, but your registration card must be on file before we will provide any additional Tech Support. You may wish to send your card to us by overnight courier if you purchased your Periscope from a dealer and need Tech Support right away.

Please be prepared to answer these questions when you call:

- What Model (I, II, II-X, III, or IV) and version (e.g., Version 5.00) of Periscope are you using?
- What brand and model of computer are you using?
- What version of DOS are you using?
- What boards are installed in your system?
- What device drivers and/or memory-resident software are you using?
- What Periscope installation options are you using?
- What is the problem you're experiencing?

## TROUBLESHOOTING

### DOS 3.2 Users

If you're using MS-DOS 3.2 (the Microsoft version, not the IBM version), be aware of a bug that zaps the INT 2 (NMI) vector. After booting from a hard disk, each time you access the hard disk, DOS corrupts the NMI vector. When you successfully access a floppy disk, the problem is corrected. Ap-

---

parently there's an uninitialized pointer in DOS that is initialized by accessing a floppy disk. If you must use this version of DOS, you should either boot from a floppy or do a **DIR A:** with a disk in drive A before the point where Periscope is loaded.

### **EGA/VGA Users**

If you're using an EGA or a VGA made by a company other than IBM, check to make sure that the display adapter is not running in emulation mode. The easiest way to do this is to use the Periscope utility program, INT. Enter **INT /D 0 3** to display the interrupt vectors zero through three. If INT 2 points to segment C000, then the EGA or VGA is using NMI, usually to emulate CGA or Hercules mode. To run any model of Periscope other than II-X, this emulation must be turned off. Check your EGA/VGA documentation for details on turning the emulation off.

If you're using an early VGA board, expect problems, especially if you're using a dual-monitor system. If you have problems, check with the VGA manufacturer regarding any ROM updates since your board was manufactured. If you're still stumped, call Tech Support.

### **Hercules Graphics Users**

If you're using a Hercules monochrome graphics board in graphics mode, note that Periscope has no built-in support for the Hercules card. You can do one of three things to use Periscope in this situation: use a Hercules color card for Periscope's output; use an alternate PC for Periscope's output; or use the public domain program HERC.COM to manually shift from graphics to text mode on entry to Periscope.

### **386MAX Users**

If you're using 386MAX by Qualitas, Inc., note the following:

- When loading Periscope in high memory, make sure you have at least 124K of contiguous memory available.
- Don't try to use the AC options with PSKEY.
- Use version 2.20 or later of 386MAX to avoid conflicts with Periscope.



---

## DOS Usage

If you try to use DOS from Periscope and get the message `DOS busy`, enter `G {0:28*4}` to set a breakpoint the next time `INT 28H` is executed. When the `CS:IP` is at the start of `INT 28H`, Periscope allows DOS to be used.

## Periscope Version and Model

The version number (Arabic number) is the release number of the software, such as 5.00. The model number (Roman numeral) identifies the hardware used with the software. To determine which version and model you are using, check the default value of watch item 7. The version number is followed by an alphabetic suffix, which indicates the model. The possible values are **X** (Model II-X), **S** (Model II), **P** (Model I, Rev 2), **M** (Model I, Rev 3 and Model I/MC), **K** (Model IV), and **H** (Model III). Avoid configuring Periscope as Model II-X. Since this model does not intercept NMI, a parity error will hang the system.

## Exception Interrupts

If your program suddenly pops into Periscope with an exception interrupt, one of two things has happened. Either an illegal instruction has been executed (indicated by exception interrupt 6) or a segment wraparound (a read or write of a word at offset `FFFFH` indicated by interrupt `0DH`) has occurred. The illegal instruction is usually caused by an unbalanced stack, such as when a routine is called as a near procedure, but returns as a far procedure. Similarly, mismatched `PUSHes` and `POP`s frequently lead to an exception interrupt.

## Break-out Switch

If your break-out switch does not work, check the following items:

- If you're using MS-DOS 3.2, see the description of the DOS bug in this appendix.
- If you're using an EGA or a VGA, see the description of EGA/VGA problems in this appendix.
- If you're using a PC's Ltd 80286 machine with a BIOS dated prior to January, 1987, you'll need to get an updated BIOS.
- Make sure that your machine supports NMI. If you're

---

using a system that has only 8 chips per 64/256 KB, such as the Tandy 1000, you're probably out of luck.

- Make sure that you haven't configured Periscope as Model II-X, since this model does not hook INT 2.
- Use INT to display the values of the interrupt vectors. INTs 1, 2, and 3 should have the same segment, with offsets ascending by 5.
- Try using CLEARNMI to periodically clear out any blockage of the break-out switch.

### **Symbols and Source Code**

If you have problems displaying source code, type /L while in Periscope. If you get the message `No lines found`, check your compile and link options. If the `DOS busy` error is displayed, no source code will be displayed. If this doesn't solve the problem, check the items listed under the `disassemble source (US)` command. Do not expect to be able to access source code when debugging a TSR program (unless you're using the Periscope utility `WAITING.COM`), since DOS is usually busy.

If you have problems accessing symbols, press Alt-I to display the symbols available. If the expected symbol is not found, check the compile and link options used to generate the program. Also, if you configured Periscope to preserve the case of symbols, be sure that the symbols are entered in the same case as shown by the Alt-I display.

### **Unexpected Entry into Periscope**

If Periscope comes up unexpectedly, enter Q to display the entry reason. See the description of the Quit command in Chapter 15 for more information.

### **Periscope III Users**

If Periscope fails to load, recheck your installation and run `PS3TEST` and `PSTEST/3` to confirm the proper operation of the board. See the separate addendum on Model III hardware installation and operation for more information.

### **Periscope IV Users**

If Periscope fails to load, recheck your installation and run `PS4TEST` to confirm the proper operation of the board. See Chapters 7 and 10 for more information.



---

## **Interrupt Vectors**

Since Periscope temporarily replaces the values of some interrupt vectors while its screen is displayed, most display commands will show the values of Periscope's vectors. To see your program's vectors, use the doubleword format, `DD`. The values followed by an asterisk indicate your program's vectors.

## **Disassembly Window**

If the reverse video bounce bar disappears, enter `R` to force it to reappear.

## **Program Screen**

If your program's screen is not being correctly restored after Periscope's screen is displayed, check to be sure you've allocated enough memory for saving and restoring screens. In a single-monitor system, the default is 4KB, which is enough for text mode only. If your program uses graphics mode, we strongly recommend use of a dual-monitor system. It will save much wear and tear on you and your display adapter.

An EGA or a VGA driving a color display can co-exist with a monochrome display. Be sure to use a plain Jane display adapter, i.e., one without any graphics ability. You can use one of the original IBM mono cards or the Dell mono card or equivalent.

## **Other Problems**

If you encounter problems other than the ones mentioned above, check to see if the situation is repeatable. First, try it on the same machine under the same circumstances. If the problem persists, try removing resident programs and device drivers. If the problem goes away, try to isolate the conflicting program. Finally, try to repeat the problem on another machine. The more information you have when you call for Tech Support, the better, since that will help us help you more quickly and effectively. ♦



# Command Parameters

- **Command Parameters**
- **Character Sequences**

**T**his chapter covers the command parameters for the commands in Chapter 15 and in the Model III Addendum. It also covers character sequences that you can use to execute pre-defined command sequences (aliases).



---

## COMMAND PARAMETERS

Periscope commands 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 the command is only one character in length, 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. Square brackets ( [ ] ) in the command syntax are used to indicate an optional entry. (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.

\* — An asterisk entered as the first character in a Periscope command line causes the entire line to be treated as a comment.

? — A question mark is used to indicate a variable.

| — A vertical rule is used to indicate a logical OR.

\$ — 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.

[ ] — 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 {0:10*4}`. This command uses the offset at 0:40H and the segment at 0:42H, which is interrupt vector 10H.

**O.** — A prefix of **O.** before a symbol name extracts the offset portion of a symbol name. For example, to set AX to the offset portion of the symbol NEWPAGE, use `R AX O.NEWPAGE.`

**S.** — A prefix of **S.** before a symbol name extracts the segment portion of a symbol name. For example, assume the symbol ARRAY points to 1234:5678. To display memory at 1234:0000, you could use `D S.ARRAY:0.`

**T.** — A prefix of **T.** indicates that a decimal number follows. For example, to display memory at offset 2000 decimal, enter `D T.2000.`

**W.** — A prefix of **w.** extracts the word at the target address and uses it as the argument. For example, to display memory at offset 1000H in the segment referenced by the symbol DOSSEG, enter `D w.DOSSEG:1000.`

**+, -, \*, /** — These arithmetic operators may be used to perform inline arithmetic. The operators + (add), - (subtract), \* (multiply), and / (divide) are evaluated from left to right. For example, `dd 0:21*4, r ip ip+1, d ss:sp-4, and u bx+si-5` are all acceptable.

**<address>** — The address of a memory location, 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 `PRINTLINE`.

**<alias>** — An alias is a two-character shorthand notation that can have an associated character string of up to 64 char-

---

acters. There are currently 38 alias names defined and eight aliases reserved for use with Periscope. (Others may be defined by you.) Aliases may be entered in a DEF file or by using the EA command.

The defined and reserved aliases are:

**MP** – The module path name for source-level debugging.

**MX** – The module extension for source-level debugging.

**X0** – The commands executed when RUN transfers control to Periscope on entry to the program.

**X1** – The commands executed on entry to Periscope.

**X2** – The commands executed after each Periscope command.

**X3** – The commands executed on exit from Periscope.

**C0** – The commands to reset Periscope's windows to their last settings.

**C9** – The commands to reset Periscope's windows to their original (default) settings.

**FX** – Defines the contents of the menu bar. Used with xxKEYS.PSD. See NOTES.TXT.

The defined aliases (which you can assign to suit your needs) are:

**A0-A9** (Alt-F10, Alt-F1 thru Alt-F9)

**C1-C8** (Ctrl-F1 thru Ctrl-F8)

**F0-F9** (F10, F1 thru F9)

**S0-S9** (Shift-F10, Shift-F1 thru Shift-F9)

**< arithmetic operator >** – The arithmetic symbols +, -, \*, and /, are 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.

**< command >** – A Periscope command, such as US (unassemble source) or D (display in current format).

**< 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. Also

---

you may use **A:**, **B:**, etc.

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

**<flag>** — A flag register. The possible values and two-character mnemonics are shown in Figure C-1.

FLAG	SET (=1)	CLEAR (=0)
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)

*Figure C-1. Flag Register Values/Mnemonics*

**<format>** — The formats available with the display command, such as **A** (ASCII), **B** (byte), etc..

**<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.

---

**NOTE:** Don't use the `L nnnn` form if you have a symbol whose name is `L`. Use an address instead.

**<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 one- to 64-character name used as an alias.

**<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.

**<pointer>** — A bracket ( `[]` ) or brace ( `{}` ) used to indicate a near or far pointer respectively.

**<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, FS, GS, 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.

**<segment>** — A one- to four-digit hexadecimal number or register representing one of the six segment registers (CS, DS, ES, FS, GS, SS).

---

**<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.

**<symbol>** — A name corresponding to an address or a record definition. Symbols are loaded from a PSS file when the corresponding program is loaded by RUN. A symbol name may be optionally preceded by a period. (If a period precedes a name, the name is forced to be a symbol.) For example, to disassemble memory starting at the symbol PRINTLINE, enter `U .PRINTLINE`.

Symbols are evaluated first, before numbers and registers. This can cause conflicts and/or confusion if a symbol has the same name as a valid register or number (e.g. AX or A123). Be careful not to confuse symbol names with addresses. The command `D A123` first tries to use A123 as a symbol name. If a symbol is not found, A123 is used as a hex number. To inhibit its use as a hex number, use `D .A123`.

A tilde (~) is used to force the name not to be treated as a symbol. For example, to display memory at address F00 when a symbol of the same name exists, enter `D ~F00`.

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>** — 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).

## CHARACTER SEQUENCES

**^F1-^F9, and ^F0** — Use to execute F1–F10.

**!CA-!CZ** — Use to execute Ctrl-A through Ctrl-Z.

**!AA-!AZ** — Use to execute Alt-A through Alt-Z.

---

**!A1-!A9** – Use to execute Alt-1 through Alt-9. ♦



---

# Index

!A1-!A9 318  
!AA-!AZ 317  
!CA-!CZ 317  
ag command parameter 315  
\$ command parameter 312  
\* command parameter 312  
286/386 Pod 55  
386 pod rev 1 55  
386MAX 31, 84, 162, 304, 307  
43-line mode 75  
50-line mode 74 - 75  
80286 58, 127, 131, 203  
80386 58, 127, 131 - 132, 204 - 205, 240  
    Control programs 31, 84  
    Debug registers 31, 161 - 162  
    Register display 139  
80486  
    Debug registers 31  
8086 81  
8087 43, 51  
? command parameter 312  
[] (brackets) command parameter 312  
^F1-^F9, and ^F0 317  
^XX 184  
{ } (braces) command parameter 313  
| command parameter 312

## A

Absolute address 264  
Active remote mode 74  
Address 284  
address command parameter 313  
Addressing the Model I board 38  
alias command parameter 313  
Aliases 15, 19, 107, 110, 139, 184, 230, 255, 261, 283  
    A0-A9 314  
    C0 111, 314  
    C1-C8 314  
    C9 111, 314  
    Chained 109  
    Defined 109  
    F0-F9 314  
    Format 110

Fx 111, 314  
MP 110, 184, 314  
MX 110, 184, 314  
Reserved 110  
S0-S9 314  
X0 111, 184, 314  
X1 111, 184, 314  
X2 111, 184, 314  
X3 111, 184, 314  
Alternate keyboard 76, 80, 99  
Alternate monitor 75  
Alternate PC 76, 99, 104  
Alternate start-up methods 86  
Alternate video 76, 99  
AND mode 158  
arithmetic operator command parameter 314  
ASCII 173 - 174, 264  
Assembly mode 252  
Asterisk (\*) 175  
Augat chip puller 61  
AUTOEXEC.BAT 87

## B

Binary 264  
BIOS 78, 84, 124  
Bit mask 129  
Boot 235  
Borland 92, 121  
Both (source and assembly) mode 252  
Brackets [] 312  
Break-out switch 6, 102, 235, 308  
Breakpoint overrun 132, 193, 207  
Breakpoint types 152  
    Code 152  
    Debug register 153  
    Hardware 153  
    Monitor 152  
Breakpoints 112, 128  
    Code 187, 190, 192, 194, 235  
    Debug register 187, 190, 192, 194, 235  
    Hardware 161, 192 - 193, 235  
    Monitor 190 - 191, 194, 235  
BS-16 memory 204  
Buffer capture 198

Bus compatibility 131  
byte command parameter 314

## C

Call tracing 140  
Capture keystrokes 268  
Case sensitivity 32  
CED 138  
CEMM 31, 162, 304  
CGA 83  
Chips and Technologies 41  
Clear breakpoint 154  
Clear Periscope's screen 228  
CLEARNMI.COM 102  
CMOS 30  
Code breakpoints 191  
    Sticky 188  
    Temporary 188  
Code timing 142  
CodeView 33, 120  
Color attribute 78, 266  
Colors 266  
COM file 111 - 113  
command command parameter 314  
Command delimiter 312  
Command length 31  
Command parameters 312  
Command types  
    Controls 150  
    Disassembly 150  
    Disk I/O 150  
    Display 150  
    Execution 150  
    Hardware 150  
    I/O 151  
    Miscellaneous 151  
    Option 151  
    Search 151  
    Status 151  
    Symbols 151  
Commands 150  
    /M 96  
    Assemble then Unassemble (AU) 157  
    Assemble to memory (A) 156  
    Breakpoint on 80386 Debug registers (BD)  
161 - 162  
    Breakpoint on Byte (BB) 159  
    Breakpoint on Code (BC) 160  
    Breakpoint on eXit (BX) 170  
    Breakpoint on Flag (BF) 163  
    Breakpoint on Interrupt (BI) 163  
    Breakpoint on Line (BL) 164  
    Breakpoint on Memory (BM) 165  
    Breakpoint on Port (BP) 166  
    Breakpoint on Register (BR) 167  
    Breakpoint on User test (BU) 168  
    Breakpoint on Word (BW) 169  
    Clear (K) 228  
    Clear and Initialize 228  
    Compare (C) 171  
    Display Effective address (DE) 175  
    Display using ASCII format (DA) 173  
    Display using asciiZ format (DZ) 182  
    Display using Byte format (DB) 174  
    Display using current format (D) 172  
    Display using Double word format (DD) 176  
    Display using Integer format (DI) 176  
    Display using long integer format (DX) 1  
    Display using Long real format (DL) 177  
    Display using long signed integer format (182  
    Display using Number format (DN) 177  
    Display using Record format (DR) 178 - 1  
    Display using Short real format (DS) 180  
    Display using Word format (DW) 180  
    Enter (E) 183  
    Enter Alias (EA) 184  
    Enter Bytes (EB) 185  
    Enter Doublewords (ED) 185  
    Enter Symbol (ES) 185  
    Enter Words (EW) 186  
    Fill (F) 187  
    Go (G) 187 - 188  
    Go equal (G =) 190  
    Go plus (G +) 189  
    Go to Return address on stack (GR) 193  
    Go using All (GA) 190  
    Go using Monitor (GM) 191 - 192  
    Go using Trace (GT) 194  
    Help (?) 156  
    Hex arithmetic (H) 195  
    Input (I) 224  
    Interrupt Compare (IC) 224  
    Interrupt Restore (IR) 225  
    Interrupt Save (IS) 226  
    Jump (J) 226  
    Jump Line (JL) 227  
    Load Absolute disk sectors 229  
    Load alias and record Definitions (LD) 2  
    Load Batch file (LB) 229  
    Load File from disk (LF) 231  
    Load Symbols from disk (LS) 231  
    Move (M) 232  
    Multiple 144  
    Name 233  
    Option A (/A) 266  
    Option C (/C) 266  
    Option D (/D) 267  
    Option ditto (/") 265  
    Option E (/E) 267  
    Option K (/K) 268

- Option L (/L) 269
  - Option M (/M) 270
  - Option N (/N) 270
  - Option Q (/Q) 271
  - Option R (/R) 271
  - Option S (/S) 272
  - Option T (/T) 273
  - Option U (/U) 274
  - Option W (/W) 274 - 278
  - Option X (/X) 279 - 280
  - Options 1 and 2 (/1 and /2) 265
  - Output (O) 234
  - Quit (Q) 234 - 236
  - Register 237 - 240
  - Register Compare (RC) 241
  - Register Restore (RR) 242
  - Register Save (RS) 242
  - Search for Address Reference (SA) 244
  - Search for Calls (SC) 245
  - Search for Return address 246
  - Search for Unassembly match (SU) 247
  - Search then Display 246
  - software Breakpoints All (BA) 158
  - Trace 248
  - Trace all but Interrupts (TI) 251
  - Trace Back (TB) 249 - 250
  - Trace Line (TL) 252
  - Trace Registers (TR) 249 - 250
  - Trace Unasm 249 - 250
  - Translate (X) 263
  - Use 16 or 32-bit disassembly (16 or 32) 264
  - View file (V) 257
  - View Source file (VS) 258
  - Watch 259
  - Write Absolute disk sectors (WA) 260
  - Write alias and record Definitions (WD) 261
  - Write Batch file (WB) 261
  - Write File to disk (WF) 262
  - Write Symbols to disk (WS) 263
  - Common problems 305
    - Displaying source 309
    - Displaying symbols 309
    - Model III fails to load 309
    - Model IV fails to load 309
    - No bounce bar 310
    - No program interrupt vectors 310
    - Program screen not restored 310
    - Unexpected entry into Periscope 309
  - Compaq
    - 386 82
    - 80386 DeskPro 162
  - Compatibility 9
  - Continue 235
  - Conventions 5
  - Copy memory 232
  - Corruption of low memory 115
  - CPU 58, 203
  - CPU cycle count 130 - 131, 212, 215
    - capture 199
  - CPU event 203
  - Crowbar tool 56
- ## D
- Data breakpoints 129, 132
  - Data window 267, 276
    - Select 267
  - Debug register breakpoints 191
  - Debug user exits 125
  - Decimal 264
  - Decimal number 313
  - decimal number command parameter 314
  - DEF file 19, 82, 107, 113, 179
  - DeSmet 120
  - Device driver 118
  - DIP switch 36, 54, 81, 107
  - Disable breakpoint 154
  - Disassemble memory 252
  - Disassembly
    - Modes 17, 20
    - Window 277, 310
  - DMA signals 128, 131
  - DOS 82, 112 - 114, 125, 266, 308
  - DOS busy 287
  - Double word format 175
  - drive command parameter 314
  - Dual-monitor system 75, 83, 95, 99, 310
  - Dumb terminal 76 - 77
- ## E
- Echo screen to file 267
  - Effective address 237 - 238
  - EGA 75, 83, 95, 99, 307
  - Ellipsis 312
  - Enable breakpoint 154
  - EOI 98
  - Error messages 284
  - Exception interrupts 130, 235, 308
  - Exclude state 201
  - EXE file 90, 111 - 113, 120, 262
  - EXEC function 114
  - Exit to DOS 279
  - Extended memory 215 - 216
- ## F
- Far pointer 109, 313

Fast color output 31  
FCB 112  
FCC compliance 5  
file command parameter 315  
Files on distribution disk 8  
Fill a block of memory 187  
Filter 217  
Filter function 121  
Filter leading character 121  
Flags 237  
Flex cable 55, 63  
format command parameter 315  
FTOC.C 14  
FTOC.DEF 14  
FTOC.EXE 14  
FTOC.MAP 14  
Function key emulation 33

## G

Go Using Hardware (HC) Command 191  
Grrr! 32

## H

Hardware breakpoints 128, 191  
    Data bit mask 128  
    Data values 128  
    I/O port access 128  
    Memory access 128  
    Pass counter 128  
    Selective capture 128  
    Sequential triggers 128  
Hardware commands  
    HA 196  
    HB 197  
    HC 198 - 201  
    HD 202 - 205  
    HM 206 - 208  
    HP 209  
    HR 210 - 217  
    HS 218  
    HT 218 - 219  
    HU 220 - 222  
    HW 223  
Hardware interrupts 98  
Hardware requirements 8  
Hardware Trace Buffer 78, 210 - 219  
Hardware updates 2  
Hardware-assisted debugger 127  
Hardware/software combination breakpoint  
    129  
HERC.COM 85, 307

Hercules 85, 307  
Hex 174, 196, 264  
High-level languages 13  
Hot keys 103, 235

## I

I/O Ports 36 - 37, 54, 81, 224, 234  
IBM  
    PS/2 32, 199  
Illegal instructions 239  
Initialize 228  
Installation options 74  
    /2 74  
    /25 74  
    /286 75  
    /386 75  
    /43 75  
    /50 75  
    /A 75, 142  
    /AD 142  
    /AK 76  
    /AV 76, 142  
    /B 77  
    /C 78  
    /D 78  
    /E 78  
    /H 79  
    /I 80  
    /K 80  
    /L 80  
    /M 81  
    /N 81  
    /P 81  
    /Q 82  
    /R 82  
    /S 83  
    /T 83  
    /V 84  
    /W 85  
    /Y 85  
    ? 74  
    Format 74  
INT.COM 102  
Interrupt Comments 79  
Interrupt request lines 80  
Interrupt trace table 190  
Interrupt vectors 84, 102  
Interrupts 175, 224 - 226  
IRQ 80, 98

**J**

Jumpers 68

**K**

Keyboard translation programs 85

Keyboard usage

- Alt-3 keys 139
- Alt-A keys 139
- Alt-B keys 140
- Alt-C keys 140
- Alt-D 140
- Alt-E keys 140
- Alt-G keys 140
- Alt-H keys 140
- Alt-I keys 141
- Alt-L keys 141
- Alt-M keys 142
- Alt-N keys 142
- Alt-O keys 142
- Alt-P keys 142
- Alt-R keys 142
- Alt-S keys 142
- Alt-T keys 142
- Alt-W keys 143
- Backspace key 138
- Ctrl-B keys 143
- Ctrl-Break keys 143
- Ctrl-End keys 138
- Ctrl-G keys 143
- Ctrl-Left keys 138
- Ctrl-P keys 143
- Ctrl-PgDn keys 138
- Ctrl-PgUp keys 138
- Ctrl-PrtSc keys 143
- Ctrl-R keys 143
- Ctrl-Right keys 138
- Ctrl-S keys 144
- Del key 138
- Down Arrow key 138
- End key 138
- Esc key 138
- Home key 138
- Ins key 138
- Left Arrow key 138
- PadMinus key 144
- PadPlus key 144
- PgDn key 144
- PgUp key 144
- Right Arrow key 138
- Semi-colon key 144
- Shift-PrtSc keys 144
- Tab key 138
- Up Arrow key 138

**L**

LCC (Leaderless Chip Carrier) 58

Length argument 285

length command parameter 315

Line numbers 91

Line-number symbols 90

LINK 92

LINK4 92

LINK86 120

List 285

list command parameter 316

Loading symbol tables 117

Local code 90

Local data 90

Local symbols 90

Local variables 18

Long boot 235

Long real (quad word) format 177

**M**

MAP file 14, 83, 90, 119 - 120

Memory breakpoints 128

Code prefetch 206

CPU halt 206

Interrupt acknowledge 206

Read 206

Write 206

Memory conflicts 36

Memory-resident program 98

Menu bar 111, 314

Menu system 33, 79

Messages 282

\*Break\* 282

Breakpoint cleared 282

DOS 3.00 or later required 282

DOS busy 269

EOI issued for IRQ x 282

Exception Int x 282

Grrr! 282

Keyboard interrupts (IRQ 1) are turned off...

283

Lines found 269

No buffer found 269

No lines found 269

Parity error 1 282

Parity error 2 282

Press Esc to end full-screen mode 283

Source buffer too small 283

Source file 283

Source file more recent than program 255, 283

Timer interrupts (IRQ 0) are turned off... 283

Microsoft 92, 120

Windows 5

- Microsoft C 15
- Microsoft LINK 15
- Microsoft Windows 95, 122
  - Messages 270
  - Version 3.0 95
- Mixed (Both) mode 113
- MODE 85
- Model II Break-out Switch 48 - 52
- Modify flags 239
- Modify memory 183, 185 - 186
- Modify registers 239
- Module name 255
- Money-back guarantee 4
- Monitor breakpoints 191
- MS-DOS 3.2 306

## N

- name command parameter 316
- Naming files 233
- Near pointer 109, 312
- Nearest symbols 270
- Network card 36, 54
- NMI 31, 103, 128, 132, 282, 306 - 307
- non-DOS programs 80, 97
- Non-Maskable Interrupt (NMI) 50, 102
- NOTES.TXT 8
- Nul-terminated ASCII format 182
- Null-modem cable 76 - 77, 104
- Number 285
- number command parameter 316
- Numeric co-processor 124

## O

- O. command parameter 313
- Octal 264
- Offset 313
- offset command parameter 316
- On-line help 79
- OPTASM 244
- OPTLINK 92
- Overwriting low memory 130

## P

- Parity error 43, 51
- Parity error 1 235
- Parity error 2 235
- Pass count 198
- Pass counter 129

- Passive remote mode 75
- Pause modes 140
- Periscope II break-out switch 283
- Periscope IV 54, 127
  - Compatibility 131
  - Power Requirements 133
- Periscope model differences
  - All models 8
  - Model I 6, 81
  - Model I/MC 6, 81 - 82
  - Model II 7, 81 - 82
  - Model II-X 7, 81 - 82, 103
  - Model III 7, 192
  - Model IV 7, 192
- Periscope version and model 308
- Periscope's windows 85, 112
- Periscope/Remote 74
- Peter Norton's TS.EXE 113
- PGA (Pin Grid Array) 58
- PLCC (Plastic Leaderless Chip Carrier) 58
- PLINK 93, 122
- PLINK overlays 94
- Plus Board 40, 128
- PocketSoft 93, 122
- Pod 55
- pointer command parameter 316
- Port breakpoints 128
- port command parameter 316
- Port conflicts 36
- Port read 209
- Port write 209
- Pre-DOS programs 97
- Prefetch queue 215
- Printer 142, 144
- Printer echo 143
- Printer spacing 140
- Probe 212
- Probe triggering 199
- Program loader 111
- Protected memory 105
- Prototype card 36, 54
- PS.DEF 179
- PS4TEST 223
- PS4TEST.EXE 106
- PSB file 268
- PSBUF.DAT 106, 223
- PSD file 96, 107, 113, 230, 261
- PSF file 121
- PSHELP.TXT 79
- PSINT.TXT 79
- PSKEY.COM 95, 103, 235
- PSP 112
- PSS file 113, 119, 232, 286
- PSTERM.COM 77, 104
- PSTEST.COM 105
- Public symbols 90

## Q

QEMM.SYS 31, 84, 304  
 QEXT.SYS 84  
 Qualitas 162  
 Quiet 271

## R

Replacement 3  
 range command parameter 316  
 Real-time trace buffer 7, 78, 128, 130, 192, 210 - 219  
 Record definitions 14, 19, 82, 107 - 108, 112, 140, 178, 230, 261, 317  
   File 113  
   Format 108  
 Refresh interrupts 32  
 Register 285  
 register command parameter 316  
 Register display 142  
 Register window 276  
 Registers 237  
   User 237  
 Registration 2  
   Card 306  
   Number 306  
 Remote debugging 7  
 Remove file extension 122  
 Remove path 122  
 Removing Periscope from memory 85  
 Response file 87, 292  
 Restore debug settings 261  
 Return authorization 4  
 Return to DOS 236  
 Ribbon cable 55  
 ROM-scan time 82  
 RS 83, 96  
 RS.COM 107 - 110  
 .RTLlink 93, 121  
 RUN 20, 114  
 RUN.COM 15, 111 - 114

## S

S. command parameter 313  
 Sage 93  
 SAMPLE.ASM 19  
 SAMPLE.COM 19  
 SAMPLE.MAP 19  
 Save debug settings 261  
 Screen buffer 83, 112  
 Screen swap 142

Search 216 - 217  
 Search for symbols 270  
 Search memory 243 - 244, 246 - 247  
 Search stack 245 - 246  
 sectors command parameter 316  
 Segment 284, 313  
 Segment change 272  
 segment command parameter 316  
 Selective capture 129, 200  
 Sequential triggers 129, 206 - 207  
 Serial port 76  
 Serial printer 85  
 Set breakpoint 154  
 SETUP.COM 30  
 Shadow RAM 82  
 Shift-PrtSc 103  
 Short boot 78, 80, 97, 236  
 Short real (double word) format 180  
 SIDEKICK 195  
 Signed integer (word) format 177  
 Single step 190, 194, 248, 251  
 Single-monitor system 83, 310  
 SKIP21.COM 115 - 116  
 SLR 92  
 SLR Systems 244  
 Software installation options  
   See Installation options  
 Software interrupts 190  
 Software pass count 160  
 Software requirements 8  
 Software trace buffer 77, 112  
 Source code 18, 255  
 Source File Buffer 78, 112  
 Source file name 255  
 Source only mode 252  
 Source-code line 227, 255  
 Source-level debugging 13, 90, 93, 125, 258  
 Source-only mode 113  
 Split-screen mode 74  
 Stack data 90  
 Stack display 142  
 Stack window 277  
 State machine 129  
 Static variables 122  
 Sticky code breakpoints 21, 152  
 string command parameter 317  
 String search 257  
 Supported compilers 91  
 Supported linkers 91  
 Suppress Periscope's screen output 271  
 Switch SW1 36  
 Switch SW2 37  
 Switch symbol tables 265  
 SYM file 120  
 symbol command parameter 317  
 Symbol file 113, 119  
 Symbol table 83, 112, 117, 263, 265

- Global segment changes 272
- Symbols 21, 90, 141, 157, 186, 232, 239, 313, 315 - 317
  - Line 272
  - Public 272
  - Removing 271
- SYMLOAD.COM 117
- SYSLOAD.SYS 118
- SysReq 103
- System bus 203
- System requirements 8 - 10

## T

- T. command parameter 313
- Tab stops 31
- Tape backup card 36, 54
- Tech support 2, 305
- Temporary code breakpoints 152
- Temporary symbol space 114
- test command parameter 317
- Testing the Model I board 105
- Tilde (~) 317
- Timeout 77
- TLINK 92
- Trace buffer
  - Hardware 249
  - Software 249
- Trace interrupt table 273
- Trace overflow stop 199
- Tracing ROM 227
- Translate 263
- Trigger location 200
- TS.COM 83, 119 - 122
- TSR 125
- Turbo card 31
- Turbo Debugger 33
- Tutorials 13

## U

- Upgrades 3
- Uninitialized pointers 115
- Unknown symbol 286
- Unsigned integer (word) format 176
- Unsigned long integer 181
- User breakpoint 124
- User exits 80, 123 - 124, 236, 274
- User-written code 123
- USEREXIT 80
- USEREXIT.ASM 123 - 124, 168, 274
- Using Model I board with Model IV 40

## V

- VGA 74 - 75, 82 - 83, 95, 99, 307
- View file 257
- View source file 258
- View window 277
- Virtual 86 mode 304

## W

- W. command parameter 313
- WAITING.COM 125
- Warranty 3
- Watch
  - Variables 21, 112, 259
  - Window 17, 259, 278
- Watch memory 259
- Watch ports 259
- Watchdog timer 32, 99, 235, 282
- Wildcard 243
- Window specification 288
- Windows 5, 18, 143, 275
  - Colors 275
  - Set up 274 - 278
- Word format 180
- Write file to disk 262
- Write Periscope symbol file 263
- Write-protected memory 6, 81, 128

## X

- xxKEYS.PSD 111, 314