

ARTISAN®
TECHNOLOGY GROUP

Your **definitive** source for quality pre-owned equipment.

Artisan Technology Group
(217) 352-9330 | sales@artisantg.com | artisantg.com

Full-service, independent repair center
with experienced engineers and technicians on staff.

We buy your excess, underutilized, and idle equipment
along with credit for buybacks and trade-ins.

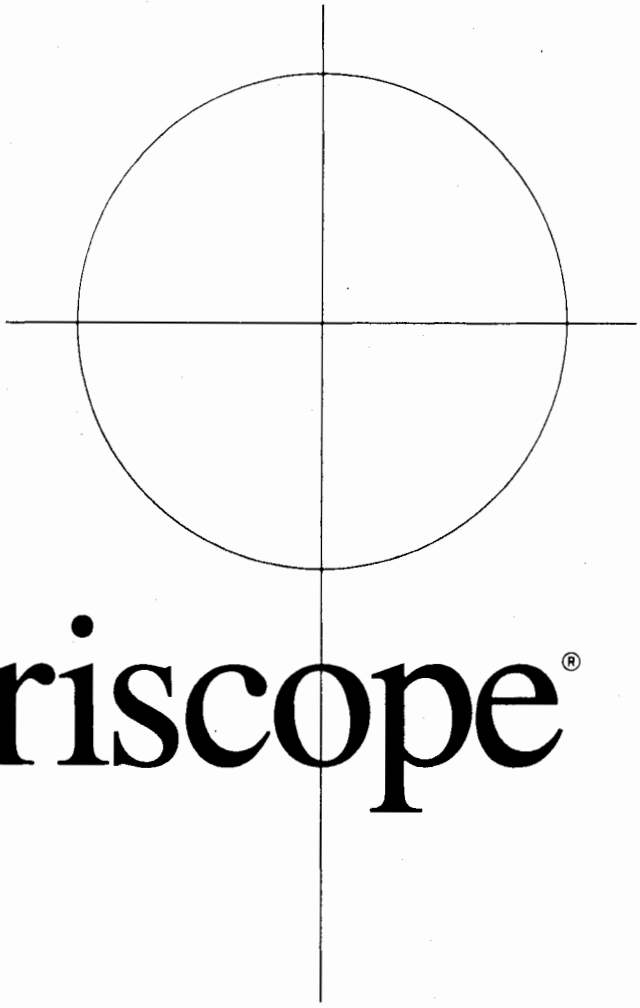
Custom engineering
so your equipment works exactly as you specify.

- Critical and expedited services
- In stock / Ready-to-ship
- Leasing / Rentals / Demos
- ITAR-certified secure asset solutions

Expert team | Trust guarantee | 100% satisfaction

All trademarks, brand names, and brands appearing herein are the property of their respective owners.

Visit our website - *Click HERE*



Periscope®

Professional Software and Hardware-Assisted Debuggers

Periscope Model IV Manual

Periscope Model IV Manual Version M54
License Agreement

Should you have questions concerning the Program or this License Agreement, please contact:

The Periscope (Computing) Co., Inc., 1475 Peachtree St., Suite 100, Atlanta, GA 30309 USA
Phone: 404/888-5335 **FAX:** 404/888-5520 **Sales:** 800/722-7006 (US & Canada)
Support: 404/888-5550 **BBS:** 404/888-5522 **Email:** 72662.3542 @Compuserve.com

LICENSE. The Periscope Computing Company, Inc. ("TPC") grants you a limited, non-exclusive license ("License") in the specified version of TPC's software product identified in this manual ("Program") to (i) install and operate the copy of the Program in machine-executable form on one computer at a time and (ii) make one archival copy of the Program. TPC and its third party suppliers retain all rights to the Program not expressly granted in this Agreement.

OWNERSHIP OF PROGRAM AND COPIES. This license is not a sale of the original Program or any copies. TPC and its third party suppliers retain the ownership of the Program and all subsequent copies of the Program made by you, regardless of the form in which the copies may exist. The Program and accompanying manual(s) ("Documentation") are copyrighted works of authorship and contain valuable trade secrets and confidential information proprietary to TPC and its third party suppliers. You agree to exercise reasonable efforts to protect the proprietary interest of TPC and its third party suppliers in the Program and Documentation and maintain them in strict confidence.

USER RESTRICTIONS. You may physically transfer the Program from one computer to another provided that the Program is operated only on one computer at a time. You may not electronically transfer the program or operate it in a time-sharing or service bureau operation. You agree not to translate, modify, adapt, disassemble, decompile, or reverse engineer the Program, or create derivative works based on the Program or Documentation or any portion thereof. You may not reproduce or distribute the Documentation or any part thereof without the prior written consent of TPC.

TRANSFER. You may not rent, lease, sublicense, sell, assign, pledge, or transfer or otherwise dispose of the Program or Documentation, on a temporary or permanent basis, without the prior written consent of TPC, except you may transfer the Program and Documentation to another party so long as that party agrees to this License Agreement and you retain no copies of the Program or Documentation.

LIMITATION OF WARRANTY AND LIABILITY. TPC warrants that the Program media and the Documentation provided herein are free of defects in materials or workmanship, assuming normal use, for a period of ninety (90) days from the date of purchase by you as evidenced by a copy of your receipt. In the event a defect occurs in materials or workmanship within the warranty period, TPC will replace the defective item(s). TPC AND ITS THIRD PARTY SUPPLIERS MAKE NO OTHER WARRANTIES OR REPRESENTATIONS, EXPRESS OR IMPLIED, REGARDING THE PROGRAM, MEDIA OR DOCUMENTATION AND HEREBY EXPRESSLY DISCLAIM THE WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. TPC and its third party suppliers do not warrant the Program will meet your requirements or that its operation will be uninterrupted or error-free. TPC, its third party suppliers, or anyone involved in the creation or delivery of the Program or Documentation to you shall have no liability to you or any third party for special, incidental, or consequential damages (including, but not limited to, loss of profits or savings, downtime, damage to or replacement of equipment and property, or recovery or replacement of programs or data) arising from claims based in warranty, contract, tort (including negligence), strict tort, or otherwise even if TPC or its third party suppliers have been advised of the possibility of such claim of damage. The liability of TPC and its third party suppliers for direct damages shall not exceed the actual amount paid for this copy of the program. Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitations or exclusions may not apply to you.

U.S. GOVERNMENT RESTRICTED RIGHTS. The Program and Documentation are provided with RESTRICTED RIGHTS. Use, duplication, or disclosure by the Government is subject to restrictions as set forth in subdivision (c)(1)(ii) of the Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subdivision 9(c)(1) and (2) of the Commercial Computer Software--Restricted Rights 48 CFR 52.227-19, as applicable. Contractor/manufacturer is The Periscope Computing Company, Inc., 1475 Peachtree Street, Suite 100, Atlanta, Georgia 30309.

GENERAL. This License shall be governed and construed in accordance with the laws of the State of Georgia.

Table of Contents

1 Introduction 1

1.1 Using this Manual 2
1.2 What's in the Model IV package 3
1.3 Model IV Requirements and Compatibility 4
1.4 Warranties, Guarantees, Upgrades and Support 5

2 Periscope Model IV Overview 7

2.1 Capabilities 8
2.1.1 Hardware Breakpoints 8
2.1.2 Hardware Trace Buffer 10
2.1.3 Break-out switch 11
2.1.4 Write-protected Memory 11
2.2 Software Options: Periscope/EM, Periscope/32 and PopUp Periscope 12
2.3 Remote Options: Passive and Active Remote Modes 12
2.3.1 Passive Remote Mode 13
2.3.2 Active Remote Mode 14
2.3.3 Protect Mode Support 14
2.3.4 Micro Channel Support 15

3 Installing Model IV Hardware 17

3.1 System Requirements 18
3.2 Installation Overview 18
3.3 Installing the Hardware 19
3.3.1 Checking for conflicts 20
3.3.2 Setting the DIP switch 22
3.3.3 Tools you'll need 23
3.3.4 Before you begin 23
3.3.5 Get access to your target CPU 24
3.3.6 Install the pod in your target system 25
3.3.7 Install the Model IV board in your host system 30
3.3.8 (Optional) Install the break-out switch 31
3.3.9 (Optional) Install the Plus board 31
3.3.10 Make sure it works 31
3.3.11 Complete the installation 34

4 Model IV Command Tutorial 35

5 Model IV Reference 43
5.1 Periscope Hardware Commands 44
5.2 Hardware Commands 44
5.2.1 Command: Go using Hardware (GH) 46
5.2.2 Command: Go using Monitor (GM) 47
5.2.3 Command: Hardware breakpoints All (HA) 49
5.2.4 Command: Hardware Bit breakpoint (HB) 50
5.2.5 Command: Hardware Controls (HC) 51
5.2.6 Command: Hardware Data breakpoint (HD) 56
5.2.7 Command: Hardware Memory breakpoint (HM) 61
5.2.8 Command: Hardware Port breakpoint (HP) 65
5.2.9 Command: display Hardware trace buffer (Hx) 66
5.2.10 Command: Hardware Write (HW) 80
5.2.11 Command: toggle internal 486 cache (/4) 82

6 Tips on Using Model IV 83
6.1 Tracking Program Flow 84
6.2 Examining the Trace Buffer after a Crash 84
6.3 Debugging the Power-On Startup Tests (POST) 85
6.4 Capturing Specific Code 85
6.5 Detecting Hardware Interrupts 86
6.6 General Tips 87
6.7 Hardware Breakpoint Examples 88

Appendix A: Model IV Messages 91

Appendix B: Glossary 95

Appendix C: Hardware Reference 109
C.1 Jumpers on the Pods 110
C.2 Jumpers on the Model IV (Rev 2) Board 112
C.3 386SX Adapter 113

Index 115

List of Tables and Figures

Table ..... Page

3-1. Common Port Usage 21
3-2. Setting switch SW1 22

5-1. Data Access Types 57
5-2. CPU Events for Various Access Widths 58
5-3. Cycle Time by CPU Speed 72
5-4. Trace Buffer Sequence Ranges 72
5-5. Summary of Model IV Trace Buffer Commands 75
5-6. Trace Buffer File Size and Track Usage 80

Figure ..... Page

3-1. Crowbar Tool (general purpose) 27
3-2. Forked Crowbar Tool 27

5-1. Periscope/EM's Model IV Hardware Menu 44
5-2. Hardware Trace Buffer in Raw Format 67
5-3. Hardware Trace Buffer in Trace Format 68
5-4. Hardware Trace Buffer in Unassembly Format 69

C-1. Layout of the 486 pods 110
C-2. J5 Pin Assignment on 386 and 486 pods 111
C-3. Layout of the 386 pod 111
C-4. Layout of the Model IV, Rev 2 board 112
C-5. 386SX Adapter, Top and Side Views 113

C H A P T E R O N E

Introduction

- **Using this Manual**
- **What's in the Model IV Package**
- **Model IV Requirements and Compatibility**
- **Warranties, Guarantees, Upgrades, and Support**

Thank you for choosing Periscope Model IV. This manual will help you install the Model IV hardware and guide you in accessing Model IV's unique hardware capabilities.

1.1 USING THIS MANUAL

Purpose. The purpose of this manual is two-fold:

(1) to explain in detail to all Model IV users how to install and test all the Model IV hardware, including boards, pods, cables, adapters, clips, etc.; and

(2) to explain in detail to Model IV users who are using the Periscope/EM or Periscope/32 software how to use the Model IV hardware commands.



The hardware commands documented in this manual are built into the PopUp Periscope software as well as the Periscope/EM and Periscope/32 software. However, you may find it easier to use the PopUp Periscope menus and dialog boxes (documented in the PopUp Periscope manual) if you're using PopUp Periscope. The hardware commands are available via the Hardware menu on both the Periscope/EM and Periscope/32 menu bar.

If you're using the Periscope/EM or Periscope/32 software, you'll need the Periscope manual as well as this manual for complete information on using your Periscope Model IV. This manual covers all of the Model IV-specific capabilities of the Periscope/EM and Periscope/32 software. The Periscope manual covers all Periscope/EM and Periscope/32 software capabilities.

If you're using the PopUp Periscope software (with a host debugger, such as CodeView or Turbo Debugger) to access Periscope Model IV's hardware capabilities, you'll need the PopUp Periscope manual as well as this manual for complete information on using your Periscope Model IV.

Typography. We use a proportional font (proportionally-spaced characters) for most of the body text of this manual. To make it easier for you to determine the spacing of items you might enter into the computer, however, we use a fixed-space font whenever we specify syntax, commands, options, messages, etc.

This sentence is formatted in the normal proportional font we use for most body text.

```
This sentence is formatted in the fixed-
space font we use for "entry" items,
screen displays, messages, etc.
```



We point out warnings, exceptions, and special instructions with a note like this.

Glossary. If you run into any terms that you do not understand, please check the glossary in Appendix B.

Readme Files and Addenda. Please be sure to read the readme files on your disks as well as any printed addenda to this manual. They will contain the latest updates to the information in this manual.

1.2 WHAT'S IN THE MODEL IV PACKAGE

At a minimum, you should have these items for a complete Periscope Model IV system:

- 3.5" Diskettes (Periscope/EM, Periscope/32, or PopUp Periscope)
- This manual and the Periscope and/or PopUp Periscope manual
- Registration cards
- Break-out switch
- Model IV board
- Model IV pod

You may also need other cables, adapters, clips, software, etc., depending on your particular environment.

One board is currently available: the 33MHz Model IV Rev 2 board with a 16K CPU-event trace buffer. We no longer produce the 16MHz, 20MHz, or 25 MHz Rev 1 boards with a 2K trace buffer, or the 33MHz Rev 2 board with a 4K trace buffer, but we do support them.

Two pods are currently available. The 386 pod works in 386DX machines running up to 33MHz. It works (with an adapter) in 386SX machines running up to 25MHz. The 486 pod works in 486DX machines running up to 33MHz, including machines with Intel's 486DX2 and 486DX4 chips, and machines using Intel's 486SX and 486SX2 pin grid array chips.



The *Embedded Periscope* products support the 386EX and 186/188 embedded processors as well as those listed above. Please call for details.

If you think you are missing any item or have an incorrect item, please contact your dealer or call us for assistance.

1.3 MODEL IV REQUIREMENTS AND COMPATIBILITY

Single System Requirements. To run Model IV hardware in a single system, you'll need a machine with an 80386SX CPU running up to 25MHz, or an 80386DX, 80486DX, 80486DX2, 80486DX4, 80486SX (PGA style), or 80486SX2 CPU running up to 33MHz externally. Your machine must have an ISA (or EISA) bus. You'll need one full-length slot for the Model IV board and an additional full-length slot if you're using the Plus board.

Remote Requirements. To run Model IV in remote mode (see Section 2.3 for details), you'll need two 80286 or later computers (80386 or later if you're using the Periscope/32 software). The "host" computer (your "debugging station") must have an ISA or EISA bus. The "target" computer (where the software you want to debug runs) must have one of the CPUs listed above under "Single System Requirements." If you want to run Model IV in *active* remote mode (vs. *passive* remote mode), your target system must also have an ISA, EISA, or Micro Channel bus and you must connect the host and target with a null modem cable.

Software Requirements. Please see the Periscope manual or the PopUp Periscope manual for requirements on running the

Model IV software you're using in a single system. If you're running in *active remote mode,* see the Periscope manual and the appropriate addendum for your target environment (DOS, Windows, OS/2, etc.) If you're running in *passive remote mode,* install the Periscope/EM or Periscope/32 software on the host system. No debugger software runs on the target in passive remote mode.

Power Requirements. The Model IV Rev 2 board requires 16 watts of power; the 386 pod requires 3 watts of power; and the 486 pod requires 3.5 watts of power.

Compatibility. Model IV will work on most any machine with one of the specified processors so long as it can be physically installed in the machine. Most incompatible machines we've encountered have a physically inaccessible CPU chip.

The pod must be installed in a system with a 5V CPU board.

Please call Technical Support if you have compatibility questions or problems.

1.4 WARRANTIES, GUARANTEES, UPGRADES, AND SUPPORT

Warranties. We provide a one-year warranty on all new or factory-refurbished Model IV hardware to registered users. Please see the Periscope manual for details.

Guarantees. Please see the Periscope manual for details.

Upgrades. You may upgrade your registered Model IV board and Periscope software at any time. (Sorry, we do not upgrade pods, adapters, cables, or other hardware accessories.) Please call Sales for current prices and details.

Support. Registering your Model IV entitles you to Technical Support via our support line, our CompuServe forum (Go Periscope), our BBS, and/or fax. See the back of the title page of this manual for a listing of the numbers.

For additional details, please see the PopUp Periscope manual, the Periscope manual, or call us.

CHAPTER TWO

Periscope Model IV Overview

- **Capabilities**
  - Hardware Breakpoints
  - Hardware Trace Buffer
  - Break-out switch
  - Write-protected Memory
- **Software Options: Periscope/EM, Periscope/32 and PopUp Periscope**
- **Remote Options: Passive and Active Remote Modes**
  - Passive Remote Mode
  - Active Remote Mode
  - Protect Mode Support
  - Micro Channel Support

Periscope Model IV is a powerful hardware-assisted debugger for use in 80386-, and 80486-based personal computers. This chapter summarizes Model IV's hardware capabilities and describes the options you have for utilizing those capabilities in various environments.

2.1 CAPABILITIES

When you use the Model IV hardware, you have several important capabilities that you do not have when you use a software-only debugger. These capabilities include:

- The ability to universally or selectively watch for reads and writes to ranges of memory and I/O ports in *real-time* (hardware breakpoints)
- The ability to record a system history in *real-time* (hardware trace buffer)
- The ability to break into the system when it hangs (break-out switch)
- The ability to run the debugger software from private, write-protected memory (optional Plus board)

The first two capabilities are the most important and powerful ones, both because they are unique to Model IV and because they are *real-time*. This means that when you use them, your system will not run any slower than normal, so you can, for instance, capture the real-time execution of a hardware interrupt or watch a COM port without slowing down the system.

You can set software breakpoints to watch memory and I/O ports and capture a trace history with software debuggers, but when you do, your system will run slower than normal. This makes it impossible to fully debug problems in time-sensitive or real-time software with a software-only debugger.

Some software debuggers, such as Periscope/EM, enable you to set hardware breakpoints by giving you access to the real-time debug registers built into 386 and 486 chips. This capability is, however, much more limited than the hardware breakpoint capabilities of Periscope Model IV hardware.

2.1.1 Hardware Breakpoints

These are the key **hardware breakpoints** you can set with Model IV:

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

- Sequential triggers
- Selective capture

You can set **memory breakpoints** on up to eight ranges in the first 16 megabytes of system memory. Your options include memory read, memory write, code prefetch, interrupt acknowledge, and processor halt. You cannot, however, trap DMA because Model IV views the world from the perspective of the CPU, and has no access to DMA (direct memory access) signals.

You can set **port breakpoints** on up to eight ranges from zero to FFFFH. Your options include both I/O read and I/O write.

You can set **data breakpoints** on byte, word, three-byte, and doubleword values. You can use them alone or to qualify a memory or port breakpoint. You can set up to eight data breakpoints on byte ranges; specific word values; specific three-byte values, or specific doubleword values. You may also set a **bit breakpoint** to specify data values with a bit mask. Model IV supports bit values of zero, one, and 'don't care'.

You can use the **pass counter** to interrupt an executing program after a specified number of breakpoints has occurred. The default pass count is one, which interrupts execution on the first breakpoint. The value of the pass counter may be from one to FFEH (1022).

You can use **sequential triggers** to watch for a specific sequence of events in real-time. For example, you can use triggers to break on the writing of a variable from a specific subroutine only after another variable has been written. Periscope Model IV has an eight-state state machine that lets you construct up to seven levels of sequential triggers of arbitrary complexity.

You can use **selective capture** to control the type of information placed in the hardware trace buffer. Selective capturing can increase the effective depth of the hardware trace buffer significantly.

2.1.2 Hardware Trace Buffer

The hardware trace buffer is a circular buffer that captures up to 16,384 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. You can tie the probe bit to an external probe input, the CPU IRQ signal (default), or the CPU Hold Acknowledge signal by changing the jumper settings on the pod.

You can display the buffer 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 your debugger software is activated, 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 or trap, you can look in the trace buffer to see what led up to the exception.

You can 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 16K CPU events have occurred.

You can selectively capture just trigger events in the trace buffer. This is useful when you need to capture multiple widely-spaced events in real-time.

With each trace buffer entry it writes, Model IV saves the number of CPU cycles since the last trace buffer entry in the 8-bit cycle count field. Under normal conditions, this cycle

count information rarely overflows. If the cycle count does overflow, however, such as when you're using selective capture, you can force Model IV to generate cycle count overflow records in the trace buffer. This allows you to count CPU cycles between widely-spaced events. You can easily convert the CPU cycle count to time using the CPU speeds listed in Table 5-3.

You can set the trace buffer to show the events surrounding a breakpoint in three different ways:

- up 16K events *before* the breakpoint;
- up or 8K events *before* the breakpoint and 8K events *after* the breakpoint; or
- up to 16K events *after* the breakpoint.

For debugging real-time applications, you'll find these capabilities extremely valuable, since you can see how your program got where it did plus where it went afterwards, with no system slowdown!

2.1.3 Break-out switch

The break-out switch gives you the ability to break into the system anytime, even if the system has crashed and the keyboard is locked up. It generates an NMI (INT 2) signal, which activates Periscope/EM, Periscope/32, or your host debugger.

2.1.4 Write-protected Memory

The Periscope Model IV boards do not contain write-protected memory. This memory is available, however, when you use the optional Plus board with the Model IV hardware. The Plus board (which is the Model I board packaged without the software, manual or break-out switch) keeps all debugging information out of the lower 640K of DOS memory. It can run with 512K or 1MB of memory and has just a 32K footprint in the first megabyte of system memory, above the lower 640K.



A software alternative to using the Plus board to keep Periscope out of the lower 640K is to use a supporting

memory manager and run the Periscope/EM software from extended memory. See the Periscope manual or details.

2.2 SOFTWARE OPTIONS: PERISCOPE/EM, PERISCOPE/32, AND POPUP PERISCOPE

Periscope/EM and its 32-bit aware derivative, Periscope/32, are full-function standalone software debuggers that also support the Model IV hardware. PopUp Periscope adds Model IV hardware support to DOS versions of CodeView and Turbo Debugger. Periscope/EM, Periscope/32, and PopUp Periscope give you full access to the Model IV hardware capabilities.

2.3 REMOTE OPTIONS: PASSIVE AND ACTIVE REMOTE MODES

In many situations where you're using an 80386, or 80486 target system and need to debug:

(1) software running in an operating environment other than real-mode DOS, such as Windows, OS/2, or an embedded target;
(2) software running in a system with no available ISA/EISA slots;
(3) the power-on startup tests (POST),

you can use Periscope Model IV in remote mode.

Active remote mode gives you virtually all the same debugging capabilities you get when you install and use Model IV in a single system. *Passive remote mode* gives you the real-time hardware trace and trigger capabilities of Model IV with no intrusion in your target system. However, you cannot interactively debug your target system in passive remote mode nor can you stop the target system.

2.3.1 Passive Remote Mode

Using passive remote mode, you can collect and examine at will a real-time execution history of the target system **with no intrusion in the target system.** To use Model IV in passive remote mode, you install the Periscope/EM or Periscope/32 software and a Model IV board in your host system and a pod in your target system. You connect the board and pod with the 48" shielded ribbon cable.

You can use Model IV's hardware trigger capabilities to define what you want to capture in the trace buffer and to stop the host system so you can examine the trace buffer. For instance, you can trigger on writes to a specific I/O port, and capture only those writes in the trace buffer using selective capture. Or you can use sequential triggering to capture only the execution of a specific routine. Or you can set a trigger and specify that Model IV position the trigger event at the top of the trace buffer and stop collecting information when the buffer is full, so that you see only what occurred after the trigger. However, you cannot actually stop the target system.

In effect, passive remote mode gives you all the powerful and unique capabilities of the Periscope Model IV hardware, except the ability to stop the target system. You will not, however, have the normal software capability of single stepping or using any commands other than the hardware (H series) and Trace Buffer commands described in Chapter 5 of this manual. Most of these commands are available on the Periscope/EM and Periscope/32 Hardware menu.



If you have software running on the target that properly handles NMI, you can install an NMI clip so that you can stop the target system with the break- out switch or on a hardware breakpoint.

If you can create a standard symbol file on the host for the software you're debugging on the target, Periscope will display symbols and source code for the events captured in the trace buffer.

2.3.2 Active Remote Mode

When you run Model IV in active remote mode, you have full Model IV capabilities, just as you have when you install Model IV in a single system. So, unlike passive remote mode, you *can* stop the target system with breakpoints and the break-out switch, and you have full, source-level interactive debugging capabilities. (You can use the commands described in the Periscope manual as well as the hardware commands described in this manual.)

To use Model IV in active remote mode, you install the main Periscope software and a Model IV board in your host system and a pod in your target system, just as you do for passive remote mode. Unlike passive remote mode, you also install debugger software (*Periscope/Remote for DOS, Periscope/Remote for Windows, Periscope/Remote for OS/2,* or other appropriate remote software) and an NMI clip in your target system, and you connect the two systems with a null-modem cable.

The Periscope software running in the host system communicates with the remote software running in the target system, providing the full interactive, source-level debugging support. The NMI clip enables Periscope to stop the target system when a hardware breakpoint occurs or when you press the break-out switch. And the Model IV hardware provides full hardware breakpoint and trace buffer facilities.

2.3.3 Protect Mode Support

Use Model IV in passive remote mode if you need real-time debugging capabilities in a protect mode environment we do not currently support.



If you're developing your own 32-bit kernel, the Periscope 32-bit Protect Mode Toolkit enables you to create a Periscope kernel debugger for your environment. It can also provide the basis for developing a low-level debugger (ala Periscope/32 for Windows) for extensible target environments.

If you're debugging a target system with a non-DOS real mode operating system, you can license the source code for *Periscope/Remote for DOS*, then modify it to run in your target environment.

2.3.4 Micro Channel Support

You must run Model IV in remote mode to debug software running in a system with a Micro Channel (IBM PS/2) bus. Since the Model IV board is designed for an AT-compatible bus, you cannot install it in a machine with a PS/2-compatible bus. Your target operating environment determines whether you can run in active remote mode or must use passive remote mode.

CHAPTER THREE

Installing Model IV Hardware

- System Requirements
- Installation Overview
- Installing the Hardware
  - Checking for conflicts
  - Setting the DIP switch
  - Tools you'll need
  - Before you begin
  - Get access to your target CPU
  - Install the pod in your target system
  - Install the Model IV board in your host system
  - (Optional) Install the break-out switch
  - (Optional) Install the Plus board
  - Make sure it works
  - Complete the installation

You'll find the information you need to install the Periscope Model IV hardware in this chapter.

3.1 SYSTEM REQUIREMENTS

Depending on your operating environment, your host and target may or may not be one and the same system. If you are running in a single system, both the host and the target requirements described below apply to that system.

Your host system can have any 386 or higher processor. It must have an ISA/EISA slot available.

Your target must have a 386DX, 486DX, 486DX2, 486DX4, 486SX, or 486SX2 Pin Grid Array (PGA) CPU or a 386SX Plastic Quad Flat Pack (PQFP) CPU.



The *Embedded Periscope* products, which include Model IV hardware, support other embedded X86 processors as well as those listed above. Please call for details.

Periscope supports 386DX and 486DX/DX2/DX4/SX/SX2 target processors running at *external* CPU speeds (at the pins) up to 33MHz and *internal* speeds up to 100MHz, and 386SX processors running at speeds up to 25MHz.

The Periscope Model IV hardware requires a 5-volt target system. If your target CPU is a 3.3-volt 486DX4 installed on a 5-volt system board, Periscope will work. (The CPU will be plugged into an adapter rather than directly into the CPU socket if this is the case. If it is plugged directly into the CPU socket, it is a strong indication that the system board is 3.3 volts.)

Your target must have a location where the I/O Channel Check signal (NMI) or its equivalent is available. This channel is available at pin 1 of EISA and ISA busses.

3.2 INSTALLATION OVERVIEW

To get started, the first thing you'll do is install the Model IV hardware. This involves:

- making sure you have the Periscope hardware you need

- checking for conflicts between the Periscope board and your host system
- resetting the switch on the Periscope board if you have conflicts
- getting access to your target CPU and locating pin 1
- installing the Periscope pod in your target system
- installing the Periscope board in your host system
- installing the Periscope break-out switch (optional)
- installing the Periscope Plus board (optional)
- making sure it all works

You'll find detailed hardware installation instructions immediately following this overview.

Once you've installed the hardware, you'll install the debugger software, following the instructions in the appropriate manual.

3.3 INSTALLING THE HARDWARE

To install the Periscope hardware, you should have these items:

- A full-length board, usually referred to as the Model IV board, that you'll install in your host system
- A small board, referred to as the 386 pod or 486 pod, that you'll install in your target system
- A 386SX adapter that you'll install in your target if it has a 386SX CPU
- A ribbon cable that connects the Model IV board to the pod
- An NMI clip that connects the pod to the bus in the target (if you're using active remote mode)
- Tools and other items packaged with the Periscope Model IV hardware, such as the Periscope mini-Maglite, crowbar tools, and CPU spacer sockets

Socket extenders, rotator sockets, extra CPU spacer sockets, and/or chip pullers may be needed in some installations due to physical-fit issues. These items are available, so if you have problems physically installing the hardware in your system, please call Periscope Technical Support for advice.



Please refer to the Hardware Reference in Appendix C for labeled diagrams of and other information on the Model IV board, pods, and adapters.

Make sure you have the Periscope hardware you need for your debugging environment. Then, before you begin the actual hardware installation, check for any conflicts that might exist between your system and the Model IV board.

3.3.1 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 Model IV board, the DIP (Dual In-line Package) switch is set to use eight 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. Other boards may also use Periscope's default I/O ports. Check the documentation for any non-standard boards to see if this is the case. Note that the true range of I/O ports available is from zero to 3FFh, since the PC supports the ten low-order bits of a port address.

If you find conflicts with the I/O ports used by your Model IV board, you'll need to reset the DIP switch as explained below. If you find no conflicts, you can skip the next section.

I/O ADDRESS	DESCRIPTION
0000h-0019h	DMA
0020h-003Fh	8259 Programmable interrupt controller (PIC)
0040h-005Fh	Programmable interval timer
0060h	Keyboard data input/output buffer
0061h	8042 control register
0064h	8042 keyboard input buffer
0065h-006Fh	Reserved by 8042
0070h	CMOS RAM address register
0071h	CMOS RAM data register
0080h	Manufacturing test port
0081h-009Fh	DMA
00A0h	Programmable interrupt controller 2
00A1h	Programmable interrupt controller 2 mask
00C0h-00DEh	DMA
00DFh-00EFh	Reserved
00F0h-00FFh	Math coprocessor
0100h-016Fh	Reserved
0170h-0177h	Fixed disk 1
01F0h-01F7h	Fixed disk 0
01F9h-01FFh	Reserved
0200h-020Fh	Game control port
0278h-027Ah	Parallel 3
02B0h-02DFh	Reserved
02E1h	GPIB (adapter 0)
02E2h-02E3h	Data acquisition (adapter 0)
02E4h-02F7h	Reserved
02F8h-02FFh	Serial 2
0300h-031Fh	Prototype card
0320h-0324h	Fixed disk adapter
0325h-0347h	Reserved
0348h-0357h	DCA 3278
0372h-0377h	Diskette controller
0378h-037Ah	Parallel 2
0380h-038Fh	SDLC and BSC communications
0390h-0393h	Cluster (adapter 0)
03A0h-03AFh	BSC communications (primary)
03B0h-03DFh	Video - MDA, CGA, EGA, and VGA
03F0h-03F7h	Diskette controller
03F8h-03FFh	Serial 1

Table 3-1. Common Port Usage

3.3.2 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. You may set it to any I/O ports on an eight-byte boundary. Set the switch so that it does not conflict with other

	UNITS		TENS				HUNDREDS	
PORT	S1	S2	S3	S4	S5	S6	S7	S8
300	N/A	ON	ON	ON	ON	ON	OFF	OFF
308	N/A	OFF	ON	ON	ON	ON	OFF	OFF
300	N/A	ON	ON	ON	ON	ON	OFF	OFF
310	N/A	ON	OFF	ON	ON	ON	OFF	OFF
320	N/A	ON	ON	OFF	ON	ON	OFF	OFF
330	N/A	ON	OFF	OFF	ON	ON	OFF	OFF
340	N/A	ON	ON	ON	OFF	ON	OFF	OFF
350	N/A	ON	OFF	ON	OFF	ON	OFF	OFF
360	N/A	ON	ON	OFF	OFF	ON	OFF	OFF
370	N/A	ON	OFF	OFF	OFF	ON	OFF	OFF
380	N/A	ON	ON	ON	ON	OFF	OFF	OFF
390	N/A	ON	OFF	ON	ON	OFF	OFF	OFF
3A0	N/A	ON	ON	OFF	ON	OFF	OFF	OFF
3B0	N/A	ON	OFF	OFF	ON	OFF	OFF	OFF
3C0	N/A	ON	ON	ON	OFF	OFF	OFF	OFF
3D0	N/A	ON	OFF	ON	OFF	OFF	OFF	OFF
3E0	N/A	ON	ON	OFF	OFF	OFF	OFF	OFF
3F0	N/A	ON	OFF	OFF	OFF	OFF	OFF	OFF
000	N/A	ON	ON	ON	ON	ON	ON	ON
100	N/A	ON	ON	ON	ON	ON	OFF	ON
200	N/A	ON	ON	ON	ON	ON	ON	OFF
300	N/A	ON	ON	ON	ON	ON	OFF	OFF

Table 3-2. Setting Switch SW1

ports in the system. Certain ports are off-limits, such as zero to 100h and ports already in use by another board. If port 300h is not available, try 310h. Consult the technical reference manual for your system and documentation for your non-standard expansion cards to avoid conflicts. Also see the common port usage list in Table 3-1.You can read the I/O port address using Table 3-2. 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. Note that this is not a definitive list of addresses; you can theoretically use any value from 0 to 3F8. Since the board uses eight consecutive I/O ports, the I/O port address indicated by SW1 on the board must be evenly divisible by eight.



If you change the default port address, be sure to write down the new address for future reference.

3.3.3 Tools you'll need

To install the Periscope hardware, you'll need a small screwdriver as well as the tools included in your Periscope package. You may also need needle-nose pliers to straighten any bent pins you find.

3.3.4 Before you begin

Please follow the steps below in order. The installation of the hardware is fairly complex; if you skip a step or skim the instructions, you could damage the Periscope 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.



See Appendix C for labeled diagrams of the Model IV board, pods, and adapters that you may need to refer to during the installation.

Remember that if you're running in a single system, the "host" and the "target" are one and the same.

3.3.5 Get access to your target CPU

Step 1—Turn the power off and remove the power cord from your target. If applicable, open the chassis to get to your CPU board.

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.

Step 2—Locate your target's CPU chip.

If your target has an 80386DX CPU, it is usually a PGA package, so you can install a 386 pod directly, with no adapter.

If your target has an 80386SX CPU, it is most likely a surface-mounted PQFP package, which requires an adapter. You'll need to install the adapter before you install the 386 pod. If you do not have the adapter, please call us.

If your target has an 80486DX, 80486DX2. or 80486DX4, the CPU is usually a PGA package, meaning you can install a 486 pod directly, with no adapter.

If your target has an 80486SX or 80486SX2 in a PGA package, you can install a 486 pod directly, with no adapter.

Step 3—Find pin 1.

During the installation, you'll need to know which corner of your CPU is pin 1. Some CPU boards are marked, but if yours isn't, find the notched (or dotted or dimpled) corner of the CPU chip and mark the board with a felt-tip pen or use one of the adhesive "dots" in the pod package. It's not neces-

sary to mark your board if your CPU is a 386SX, since you will not be removing the CPU.

3.3.6 Install the pod in your target system

The following instructions will have you install the pod in your system (Step 4 or 5), then connect the ribbon cable and NMI clip to the pod (Steps 6 and 7). If your target is tightly packed, or if your CPU is a 386SX, it may be easier to connect the ribbon cable and NMI clip to the pod *before* you install the pod in your system.

If you're installing a pod in a target with a 386DX CPU or a 486DX, DX2, DX4, SX, or SX2 CPU, go to Step 5a now.

If you're installing a 386 pod and adapter in a target with a 386SX CPU, continue at Step 4a.

Step 4a—Install the adapter onto your CPU chip.



As you install the 386SX adapter and pod, watch out for two things:

(1) There are two different pin 1 corners on the adapter. If you do not align them correctly, you may damage the adapter, your pod, and/or your CPU.
(2) If you do not apply pressure evenly on the adapter when you're pressing it down over your CPU, you may damage the (expensive) part of the adapter that clips over the CPU.

Be sure to see the labeled diagrams of the adapters in Appendix D for reference.

Locate the round indentation or "dimple" that indicates pin 1 on your CPU chip. Line up the *notched* 'CPU pin 1' corner on the outer edge of the adapter with pin 1 of your CPU chip. Press the adapter down evenly over the chip until it sits flush with the CPU board on all sides.

Be sure to keep the adapter level while installing or removing it to avoid bending pins. Once you have aligned the adapter

correctly over the CPU, place your thumbs and forefingers on the four corners of the adapter and press down. This method helps you apply even pressure.

If you run into difficulty, do not force the adapter onto the CPU. Call Periscope Technical Support for assistance.

Step 4b—If you're using a rotator socket, install it now.

Align the rotator socket's motherboard pin 1 corner with the adapter's 386 PGA socket pin 1 corner.

Step 4c—Insert the 386 pod into the adapter or rotator socket.

If you're not using a rotator socket, line up the 'PGA socket pin 1' corner of the adapter (diagonally across from the *notched* 'CPU pin 1' corner referenced in Step 4a) with pin 1 on the 386 pod.

If you're using a rotator socket, line up the 'upgrade pin 1' corner of the rotator socket with the pin 1 corner of the 386 pod.

Press the pod into the adapter or rotator socket until it snaps into place. Double check the pin 1 alignment. Be sure that you have aligned the *notched* 'CPU pin 1' corner of the adapter with your CPU's pin 1 corner. And be sure that you have aligned the PGA socket pin 1 corner of the adapter, or the upgrade pin 1 corner of the rotator socket, with the 386 pod's pin 1 corner.



If there's any doubt in your mind about the alignment, please call Periscope Technical Support before you continue.

Once you have installed the adapter and 386 pod, go to Step 6.

Step 5a—Use the crowbar tools to remove your target CPU.

Figures 3-1 and 3-2 show the crowbar tools included with the pods.

Ground yourself, then remove your PGA CPU from its socket. Start by prying the ship up with the small end of the general-purpose crowbar tool. Work your way around the CPU, gradually loosening it. Use the forked crowbar tool for final removal. Once you have removed the CPU, check it for any bent pins. Use needle-nose pliers to carefully straighten any bent pins you find.

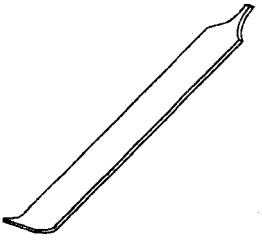


Figure 3-1. Crowbar Tool (general purpose)



Figure 3-2. Forked Crowbar Tool

Step 5b—Insert your CPU into the 386 or 486 pod.

Ground yourself, then insert your CPU into the pod after confirming that pin 1 on your CPU is aligned with pin 1 on the pod's silk-screen. Before pushing the CPU fully into position, look on the underside of the chip for any bent or misaligned pins. Do not remove the protective foam from the bottom of the pod. Place the palm of one hand over the CPU and the palm of your other hand over the protective foam, then firmly and evenly press your CPU into the socket. When your CPU is fully inserted, the exposed CPU pin length between the bottom of the CPU and the top of the pod should be about one millimeter (1/20 inch).

Step 5c—Insert the pod into your target CPU socket.

Each pod is shipped with one PGA spacer socket already installed on the bottom of the pod and one spare socket in the package. In most cases, you won't need the spare socket.



If your CPU is in a ZIF (zero insertion force) socket, drop the spare spacer socket into the ZIF socket and flip the lever to close it *before* you install the pod.

If you're using a rotator socket, install it next, aligning the rotator socket's 'motherboard pin 1' corner with your CPU board's pin 1 corner.

Remove the protective foam from the bottom of the pod. Check the pins in 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 CPU socket or with the rotator socket's 'upgrade pin 1.' Using the (way cool) Periscope Mini-Maglite as needed, insert the pod into the CPU or rotator socket. Firmly press the pod into the socket. If the pod runs into any obstructions on the CPU board, remove it and add the spare spacer socket. If you need additional spacer sockets, please call Periscope Sales.



Be very careful to correctly align pin 1 on the CPU, pin 1 on the pod, and pin 1 on the CPU board (or 'upgrade

pin 1' on the rotator socket). Failure to do so can damage the CPU chip or the CPU board.

Step 6—Plug the ribbon cable into the pod at the connector.

Use the shorter cable if you're installing both the board and the pod in the same system. Use the 48" shielded cable if you're installing the board in one system and the pod in another. Note that the cable is keyed, so that you can install it in only one position. When you remove the ribbon cable, use the crowbar tool and be careful not to bend the pins.

Step 7—If you're using active remote mode, connect the NMI clip to the pod.

Connect the pod-end of the NMI clip to the pod at pin J6. (See the pod diagrams in Appendix C.)

Step 8—If you're using active remote mode, connect the NMI clip to your target's bus.

Connect the gold-plated probe-end of the NMI clip to the I/O Channel Check signal in your target. If the target has an EISA or ISA you'll use pin A1. Pin A1 is the pin on the component (chip) side of any board, closest to its mounting bracket.

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 fingers on the board and the connector in the socket. (Not all boards have a gold finger at pin A1. Look for the first socket connector to positively identify the pin.) Push the probe in as far as possible to ensure a good connection and to keep the non-insulated part of the probe from contacting anything other than the desired pin.

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 a dummy hole at the end of the socket. Do not insert the probe in this

hole. If you cannot 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 needle-nose pliers.

3.3.7 Install the Model IV board in your host system

Step 9—Plug the ribbon cable into the Model IV board at the connector.

Note that the cable is keyed, so that you can install it in only one position. When you remove the ribbon cable, use the crowbar tool and be careful not to bend the pins.

Step 10—Install the Model IV board.

Turn the power off and open your host computer. You can install the Model IV board in any one of the available 8-bit or 16-bit full-length ISA or EISA slots.

Select an available 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 slot and lower it until the edge connector is resting on the slot receptacle. Press the board straight down until it seats in the 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 board may run into chips mounted on the system board. Make sure there's no chance of a short-circuit between the board and any other components. If you do have enough clearance under the Model IV board, we have socket extenders available. Call Periscope Sales for details.

3.3.8 (Optional) Install the break-out switch

Step 11—Install the break-out switch.

If you plan to use the break-out switch, plug it into the phono jack on the Periscope board's mounting bracket.

3.3.9 (Optional) Install the Plus board

Step 12—Install the Plus board.

Be sure the Plus board uses the same starting port as the Model IV board.

If you are going to run in remote mode, install your Plus board in your host system. If you are going to run in active remote mode using the Periscope/Remote for DOS software, you may install the board in either your host or target. Refer to the installation instructions included in the Plus board package as needed to install the board.

3.3.10 Make sure it works

Step 13—Power up your host system.

Re-connect all peripherals in your host system and turn on the power. 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 pod, since a short circuit of some sort may have occurred.

Step 14—Put your host system back together and boot it.

If you're going to run in active remote mode, refer to the instructions in the remote addendum to complete the installation after you boot your host system.

If you're running in a single system or in passive remote mode, continue with Step 15.

Step 15—Configure the Periscope software to support the Model IV hardware.

Place your distribution disk in drive A of your host system and enter **a:setup.** Follow the on-screen instructions to load the Periscope software onto your hard disk. You need to do this now so that you can run the diagnostics in the next step. You can re-run the configuration program later if you want to change any of the options.

Refer to the Periscope manual or the PopUp Periscope manual for detailed instructions.

If you're running in passive remote mode, continue at Step 17.

Step 16—Run the Model IV diagnostics.

Run the utility program PS4TEST with the appropriate options to test the breakpoints and trace buffer on the Model IV board.
PS4TEST requires that both the Model IV board and pod be installed in the same system, so if you're running in passive remote mode, go to the next step.

To run the diagnostics, enter **ps4test** followed by the appropriate options:

/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
/R	Handle RAM refresh cycles that are shown as memory reads at the CPU
/S	Set silent mode, showing only error messages
/SX	An 80386SX is in use (all memory is BS-16)

/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 disk in drive A to PSBUF.DAT
/WB	Copy the saved trace buffer from the disk in drive B to PSBUF.DAT
/X	Perform full extended memory tests

PS4TEST performs many different tests to validate the correct operation of the Model IV hardware. If no errors occur, it displays the message "No errors detected." If an error occurs, it displays a message and begins the next test.

If you get errors, please check the following items before you call Periscope Technical Support:

Is the CPU a 386DX, 386SX, 486DX, 486DX2, 486DX4, 486SX, or 486SX2? These are the processors that Model IV currently supports. (*Embedded Periscope* products support other X86 CPU's.)

Is the (external) CPU speed greater than 33MHz? As of this writing, Model IV does not support external speeds greater than 33MHz. (33MHz is the "official" maximum speed. Model IV, however, consistently passed the tests in our lab using 40MHz processors.)

Did you change the DIP switch on the Model IV board? If you changed it from the default setting, did you specify the **/P** option when you started PS4TEST?

Is there any conflict between the base I/O port used by the Model IV board and the address of any other board in your system?

Are all connections fully seated, especially the connection between the pod and the CPU socket, and the ribbon cable connection?

Step 17—Run the Plus board diagnostics.

If you installed a Plus board, follow the instructions included in your Plus board package to run the utility program PSTEST now, to test the memory on the Plus board.

3.3.11 Complete the installation

The hardware installation is now complete. Refer to the Periscope manual or PopUp Periscope manual for instructions on loading and running the debugger software.

CHAPTER FOUR

Model IV Command Tutorial

This chapter gives you practice using the Model IV hardware commands to access Model IV's hardware capabilities. If you're using the PopUp Periscope software, please see the tutorial in the PopUp Periscope manual.

This tutorial introduces you to the Model IV hardware commands that enable you to set hardware breakpoints and to control, analyze, and examine the hardware trace buffer.

Step 1—Configure Periscope for your system.

Place the Periscope distribution disk in drive A and enter:

```
A:SETUP
```

Configure Periscope as Model IV. See the instructions in the Periscope manual for details.

Step 2—Install the Periscope Software.

Enter:

```
CD\PERI
PS
```

See the Periscope manual for details.

Step 3—Display Periscope's screen.

Enter:

```
RUN
```

Step 4—Set a hardware memory breakpoint.

To set a hardware memory breakpoint on writes to the address 0:46C for a length of two bytes, enter:

```
HM 0:46C L2 W
```

Then, to execute the program with hardware breakpoints enabled, enter:

```
GH
```

You'll see the message `Setting breakpoints..` while Periscope programs the Model IV board. The DOS

screen is displayed briefly. Then Periscope's screen is displayed, along with the message `EOI issued for IRQ 0` since our breakpoint occurred during a hardware interrupt.

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

Step 5—View the hardware breakpoint event.

To see the event that caused the hardware breakpoint, enter:

```
HS
```

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 (`0000`), and the string `Bottom,` indicating that this is the last entry in the trace buffer.

Step 6—Center the breakpoint in the trace buffer.

Enter:

```
HC TC;GH
```

This sets the hardware controls (`HC`) to center the trace (`TC`) and executes with hardware breakpoints enabled (`GH`).

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.

Step 7—View the trace buffer in raw mode.

You'll quickly return to Periscope's screen, but you won't see the `EOI issued for IRQ 0` message this time, since the hardware interrupt is no longer active due to the centering of the trigger.

To turn off Periscope's windows and thus provide maximum space for the trace display, enter:

```
/W
```

You can restore the windows later using **Ctrl-F10**.

To display the trace buffer in raw mode, enter:

```
HR
```

This 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.

To switch to trace mode, press `T`.

Periscope performs an analysis of the trace buffer the first time you use this mode, to determine which prefetch instructions were actually executed. When the analysis is complete, Periscope displays `Working 100%` in the lower left-hand corner of the screen.

Periscope disassembles fetch cycles and shows other CPU events, such as memory and I/O reads and writes, interspersed in the trace mode display. If you are debugging at the source level, Periscope displays your source code when the instruction matches the beginning of a source line. From the Periscope prompt, you can get directly to trace mode by entering the `HT` command.

Step 9—View the trace buffer in unassembly mode.

To switch to unassembly mode, press `U`.

In this mode, Periscope disassembles and displays only Fetch cycles. No other CPU events are shown. From the Periscope prompt, you can get directly to this mode by entering the `HU` command.

Now press the **Esc** key to exit the trace buffer and return to the Periscope prompt.

Step 10—Use selective capture.

The Model IV trace buffer can be set up to capture just selected information.

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.

To display the trace buffer, enter:

```
HR
```

You'll see sixteen writes to 0:46C, with the value written increasing by one each time. Since there are no Fetch cycles in the buffer, the trace and unassembly modes are not useful.

Press **Esc** to return to the Periscope prompt.

Step 11—Qualify a memory breakpoint with a data breakpoint.

You can use data (`HD`) and bit (`HB`) breakpoints 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.

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 (low intensity white) 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.



The command sequence above assumes that the screen's memory is accessed a word at a time. If you have problems with this example, press the break-out switch to activate Periscope's screen, then enter `HD *;GH` to clear the data breakpoints and arm the board. After the hardware breakpoint occurs, look in the trace buffer to see how the memory was accessed and modify the `HD` command accordingly. See Section 5.2.6 for details.

Step 12—Set a port breakpoint.

To watch for the next write to port 20h, enter:

```
HA *
HP 20 O
GH
```

This command sequence clears all hardware breakpoints (`HA *`), sets a breakpoint on the next out to port 20h (`HP 20 O`), and arms the board (`GH`). When control is returned to Periscope, enter the `HS` command to display the port write.

Step 13—Use sequential triggers.

Periscope Model IV has a built-in state machine that lets you put together complex trigger conditions that it evaluates in real-time.

To stop on the first write to the timer word after the color display has been scrolled, enter the following commands:

```
HA *
HM B800:0 L2 W (0,1)
HM 0:46C L2 W (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 command sets a breakpoint on writes to 0:46C only when the board is in state 1. The exclamation point indicates a trigger condition.

Step 14—End the tutorial.

To clear all breakpoints and exit the debugger, enter:

```
HA *
QC (or G)
```

These commands will end the tutorial and take you back to the DOS prompt. You're now ready to start using your Periscope Model IV!

CHAPTER FIVE

Model IV Reference

- Periscope Hardware Menus
- Hardware Commands
  - Go Using Hardware (GH)
  - Go using Monitor (GM)
  - Hardware breakpoints All (HA)
  - Hardware Bit breakpoint (HB)
  - Hardware Controls (HC)
  - Hardware Data breakpoint (HD)
  - Hardware Memory breakpoint (HM)
  - Hardware Port breakpoint (HP)
  - display Hardware trace buffer (Hx)
    - Display Formats R, S, T, and U
    - Layout of the Trace Buffer Display
    - Analyzing and Examining the Trace Buffer
  - Hardware Write (HW)
  - toggle internal 486 cache (/4)

This chapter describes the Periscope/EM menus and commands specific to Model IV. See the Periscope manual for a summary of and details on all other Periscope/EM menus and commands. See the PopUp Periscope manual for details on the PopUp Periscope menus. The commands listed in this chapter are available in PopUp Periscope. However, you may prefer to use the menus documented in the PopUp Periscope manual.

5.1 PERISCOPE HARDWARE MENUS

When you use Model IV, both the Periscope/EM and the Periscope/32 software display a hardware menu option at the right end of the menu bar. (See Figure 5-1.) This hardware menu and its sub-menus will guide you through the syntax of using the Model IV hardware commands. (Note that PopUp Periscope provides a different set of menus.)

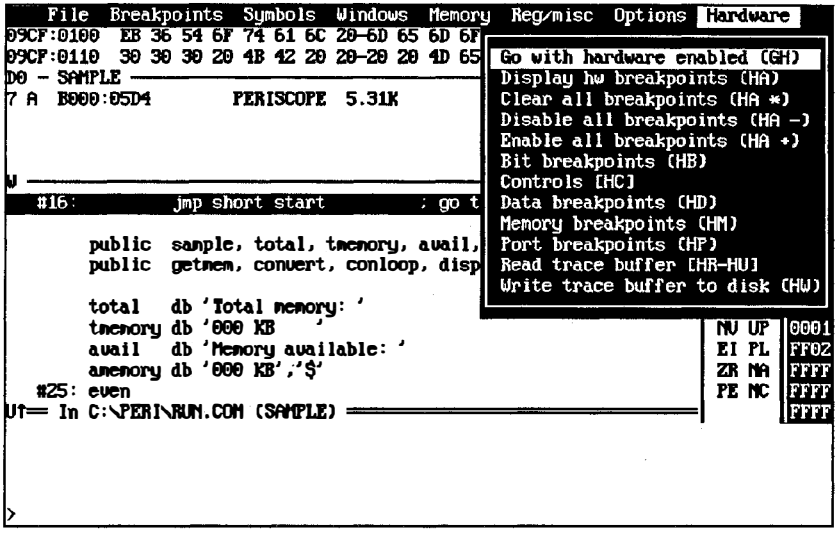


Figure 5-1. Periscope/EM's Model IV Hardware Menu

You'll see a special trace buffer commands menu when you display the hardware trace buffer.

To issue commands, you can use the menus or you can enter the commands at the Periscope command line prompt (>).

5.2 HARDWARE COMMANDS

The Model IV hardware commands (Hx) enable you to set hardware breakpoints and controls and to display and analyze the hardware trace buffer.

You set hardware breakpoints and controls with the HB (Hardware Bit), HC (Hardware Controls), HD (Hardware

Data), HM (Hardware Memory), and HP (Hardware Port) commands. You activate them with the GH (Go Hardware) and GM (Go Monitor) commands only. They run at full speed when you activate them with the GH command.



Because you can use the GM command with just the Periscope/EM software installed, you'll find it under the Periscope/EM 'Breakpoints' menu option instead of the 'Hardware' menu option. We have documented it in this manual as well as in the main Periscope manual, however, because you can use it to activate Model IV hardware breakpoints.

All hardware breakpoints are 'sticky' (remembered) until you explicitly clear them. To clear a previously-set breakpoint, just re-enter it . Periscope will display the message `Breakpoint cleared`. It's a good idea to display all breakpoints with HA ? before you enter the GH or GM command.

You display, analyze, save, and print the hardware trace buffer with the HR (display Hardware buffer in Raw mode), HS
(... in Single event mode), HT (... in Trace mode), HU (... in Unassembly mode), and HW (Hardware Write) commands. Using special commands that are available when you view the trace buffer, you can change the buffer display and search through the buffer while you view it.

You toggle the state of the internal 486 cache with the /4 command. You should turn this cache off when you use Model IV.



See the Periscope manual for an overview of the Periscope/EM menu system, for details on all Periscope/EM commands that are not specific to Model IV, and also for information on keyboard usage, command parameters, and aliases.



See the PopUp Periscope manual for information on PopUp Periscope's menus, commands, command parameters, and keyboard usage.

5.2.1 Command: Go using Hardware (GH)

Syntax:

```
GH [<address>] [...]
```

Description:

Use this command to activate code breakpoints, debug register breakpoints, and hardware breakpoints (but not monitor breakpoints), and to execute the program you're debugging. It is the same as Periscope/EM's `G` command, except that it also activates any hardware breakpoints that are enabled. (Remember that RUN disables hardware breakpoints.) Be sure to check the status of the hardware breakpoints using the `HA` command before you enter this command.



In PopUp Periscope, this command is G with no suffix or arguments.

Examples:

`GH PRINTLINE` sets a temporary code breakpoint at the address equal to the `PRINTLINE`, invokes all code and hardware breakpoints that are set and enabled, and starts execution of the program.

`GH` invokes all code and hardware breakpoints that are set and enabled and begins execution of the program.

5.2.2 Command: Go using Monitor (GM)

Syntax:

```
GM [<address>] [...]
```

Description:

Use this command to execute your program at full speed to a certain point and then evaluate monitor breakpoints. If any monitor breakpoint condition is true, Periscope displays its screen. Otherwise full speed execution of your program resumes.



This command is not available in PopUp Periscope.

This command activates code breakpoints, debug register breakpoints, and hardware breakpoints. It evaluates monitor breakpoints only after a code, debug register, or hardware breakpoint occurs. The monitor breakpoints you'll find most useful are `BB` (Byte Breakpoint), `BF` (Flag Breakpoint), `BR` (Register Breakpoint), `BW` (Word Breakpoint), and `BU` (User exit Breakpoint).

The register breakpoint is particularly powerful, since register information is not available at the hardware level. For more complex events, use the user breakpoint tests. See the Periscope manual for more information on monitor breakpoints, register breakpoints, and user breakpoints.

Be sure to check the status of the hardware breakpoints using the `HA` command before you enter this command.

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 the Periscope manual for another method of trapping writes to low memory.



You'll see a discontinuity in the hardware trace buffer each time a hardware breakpoint occurs. 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 Periscope evaluates monitor breakpoints one or more instructions after the instruction that caused the hardware breakpoint (due to breakpoint overrun), it is possible for the hardware breakpoint and the monitor (software) breakpoint to be out of sync. This can cause an occasional false or missed breakpoint. In the example above where you're watching for writes to memory when CS points to your program, code that changes CS during the breakpoint overrun interval can cause problems.

Example:

When you're using Model IV, this command acts like a combination of the `GH` and `GT` commands. Suppose you need to find where your program is writing to low memory. Since DOS and other programs can legitimately write to low memory, you need to watch for writes to low memory where the offending Code Segment points to your program. To do this, set a hardware breakpoint to watch for writes to the desired range, enter the monitor breakpoint to determine the program writing to the range, and then enter the `GM` command:

```
HM <range>
BR CS EQ CS
GM
```

Each time the write occurs in the specified range, Periscope checks to see if your program is the culprit. If so, it interrupts the program and displays its screen. If not, it resumes full speed execution.

5.2.3 Command: Hardware breakpoints All (HA)

Syntax:

`HA [?] [*] [+] [-]`

Description:

Use this command to display (?), clear (*), enable (+), or disable (-) hardware breakpoints.

If you do not set the hardware controls, HC, to other than default values, nothing is displayed for them when you enter `HA ?`.



Since RUN disables breakpoints, use this command to re-enable previously-set breakpoints.

Examples:

`HA ?` or `HA` displays all hardware breakpoints.

`HA +` enables all currently-set hardware breakpoints.

5.2.4 Command: Hardware Bit breakpoint (HB)

Syntax:

`HB [?] [*] [+] [-]`; or
`HB xB XXXX XXXX` for bytes where `x` is `L`, `M`, `N`, or `H`; or
`HB xW XXXX XXXX XXXX XXXX` for words where `x` is `L`, `M`, `H`, or blank; or
`HB x3 XXXX XXXX XXXX XXXX XXXX XXXX` where `x` is `L` or `H`; or
`HB DW XXXX XXXX XXXX XXXX XXXX XXXX XXXX XXXX` for doublewords

Description

Use this command to display (`?`), clear (`*`), enable (`+`), disable (`-`), or set the hardware data bit breakpoint.

Enter the bit breakpoint 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'. You can specify only one breakpoint, but you can it use alone or combined with a hardware memory (`HM`) or port (`HP`) breakpoint. When you combine it, the bit breakpoint is logically ANDed with the memory or port breakpoint.

See Table 5-1 on page 57 for the possible type codes.

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.

5.2.5 Command: 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:

Use this command 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).

Enter the **pass count** as a # followed by a <number> from 1 to FFEh. When the specified number of breakpoints have occurred, Periscope interrupts the executing program. If you specify 1 as the pass count, the first breakpoint will interrupt the program. If you specify 4 as the pass count, the fourth breakpoint will interrupt the program, etc. Use this pass count in conjunction with hardware breakpoints (HB, HD, HM, or HP). See the Periscope manual for information on setting pass counts on code breakpoints.

Enable **buffer capture** with HC B+ (default), disable it with HC B-, or toggle it with HC B!. This control is a master switch for buffer capture. When you disable it, no information is added to the trace buffer regardless of other settings.

Here's a situation in which you'd want to disable buffer capture. Suppose you've captured some information in the buffer but cannot save it to a disk file because DOS is busy. (Periscope/EM normally uses DOS for reading and writing disk files.) 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 you can write the trace buffer to a file. When you later view the trace buffer, enter `HC O+` to force the overflow stop on. (You can also write the trace buffer directly to a floppy disk without using DOS by entering `HW A:` or `HW B:`.)

Enable **cycle count capture** with `HC C+`, disable it with `HC C-` (default), or toggle it with `HC C!`. Use this control to force Periscope to generate CPU cycle count records in the trace buffer when the cycle count overflows. You should use this control to count CPU cycles when you've enabled selective capture with `S+`. Otherwise, leave this control in its default setting.

Enable **trace overflow stop** with `HC O+`, disable it with `HC O-` (default), or toggle it with `HC O!`. When you enable it, 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, 4K, or 16K groups (depending on which Model IV board you have), so you can examine program flow starting at a particular point.



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+` or `Hx+`, where `x` is `R`, `T`, or `U`.

Enable **probe triggering** with `HC P+` (default), disable it with `HC P-`, or toggle it with `HC P!`. Use this control to keep probe cycles from causing hardware breakpoints, and to suppress display of the probe cycles except in raw (HR) mode. When you use `HC P-`, 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.

You need this control for systems such as the IBM PS/2 Model 50, where RAM refresh cycles show up in the trace buffer. These 'garbage' cycles can be very confusing. To filter them out, set jumper J5 on the pod so that it connects pins 1 and 2. (See Appendix C for more information.) You'll see `Probe` in the trace buffer for each CPU Hold Acknowledge

signal. Then enter `HC P-` to suppress display of these `Probe` records when you view the trace buffer except in raw (`HR`) mode.

Enable **selective capture** with `HC S+`, disable it with `HC S-` (default), or toggle it with `HC S!`. When you enable it, only hardware breakpoint events are saved in the trace buffer.

Use this control with a pass count that indicates the number of events you want to occur before Periscope displays its screen. For example, if you want to capture the next 16 Outs to port 3B4h, enter `HP 3B4 O`. Then enter `HC* #10 S+` to clear the controls and set them to capture only the next 10h trigger (breakpoint) 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. You can, however, use selective capture with continuous tracing. To do this, 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.



When you use selective capture, not all CPU events are saved in the trace buffer, so it is not usually meaningful to view the trace buffer in any mode other than raw mode (`HR`).

When you enable selective capture (`HC S+`), 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 (`HC C+`).

You can set the **trigger location** to the Top, Center, or Bottom of the trace buffer using `HC TT`, `HC TC`, or `HC TB` (default) respectively.

When you set the trigger location to the Top, the trace buffer shows 16K events after the trigger event. When you set it to the Center, the trace buffer shows up to 8K events before the trigger event and 8K events after the trigger event. When you

set it to the Bottom, the trace buffer shows up to 16K events before the trigger event. `HC TC` and `HC TT` are meaningful only when you've set a hardware breakpoint, not when you press the break-out switch.



The trigger location that is in effect when a hardware breakpoint occurs controls the contents of the trace buffer. Although you can change the trigger location after a breakpoint, the contents of the trace buffer remain the same.

Use the **exclude state** to exclude data from any or all sequential trigger states from being captured in the trace buffer. Enter `HC X <byte>`, where the `<byte>` may be any state from `0` to `7`.

For example, if you enter `HC X0`, only system activity that occurs while the Model IV board is in states other than `0` is saved in the trace buffer. You'll exclude all state `0` activity.



When you set the trigger location at the top (`HC TT`) or in the center (`HC TC`) of the trace buffer, Periscope does not display its screen until the trace buffer is full. If you're also setting exclude states, you may have to set code breakpoints or use the break-out switch to activate Periscope.

To clear excluded states, enter `HC X *`.

See Section 5.2.7 on page 61 for more details on sequential triggering.

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 16K CPU events have been added to the hardware trace buffer, Periscope displays its screen.

`HC S+ #20 TC` sets selective capture on with a pass count of 20h. Since selective trace is on, Periscope ignores the `TC` command and assumes `TB` instead.

`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.

5.2.6 Command: Hardware Data breakpoint (HD)

Syntax:

`HD [?] [*] [+] [-];` or
`HD xB <byte> <byte>` for a byte range where `x` is `L`, `M`, `N`, `H`, or blank; or
`HD xW <number>` for a word breakpoint where `x` is `L`, `M`, `H`, or blank; or
`HD x3 <byte> <byte> <byte>` for a 3-byte breakpoint, where `x` is `L` or `H`; or
`HD DW <address>` for a doubleword breakpoint

Description:

Use this command to display (`?`), clear (`*`), enable (`+`), disable (`-`), or set hardware data breakpoints.

To set data breakpoints, enter a range of byte values for the byte breakpoints; a single specific value for the word breakpoints; three bytes for the three-byte breakpoints; or an address for the doubleword breakpoints. You can specify up to eight data breakpoints, and you can use this breakpoint with or without a memory (`HM`) or port (`HP`) breakpoint. If you use it with an `HM` or `HP` breakpoint, the data breakpoints are ORed together as a group and then ANDed with the memory or port breakpoints as a group.

See Table 5-1 below for the possible type codes.



Periscope handles the address you enter for a doubleword breakpoint in a special fashion. It uses the segment portion of the address as the high part of the doubleword and the offset portion of the address 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.

DATA ACCESS TYPE	TYPE CODES	BYTES USED (HIGH TO LOW)
80386SX:		
WORD	W	XX
HIGH BYTE	HB	X.
LOW BYTE	LB	.X
80386DX AND 80486DX:		
DOUBLEWORD	DW	XXXX
HIGH 3 BYTES	H3	XXX.
LOW 3 BYTES	L3	.XXX
HIGH WORD	HW	XX..
MID WORD	MW	.XX.
LOW WORD	LW	..XX
HIGH BYTE	HB	X...
NEXT BYTE	NB	.X..
MID BYTE	MB	..X.
LOW BYTE	LB	...X

Table 5-1. Data Access Types

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 way the system bus sees them. For example, the CPU sees a read of a word of 8-bit memory at B000:0 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, the CPU and the bus see a word read at an even address as one word read.

In an 80386SX 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.

ACCESS	ADDRESS (BINARY)	DATA ACCESS TYPE & START ADDRESS
80386DX AND 80486DX:		
BYTE	XXX00	LOW BYTE AT 0
BYTE	XXX01	MID BYTE AT 1
BYTE	XXX10	NEXT BYTE AT 2
BYTE	XXX11	HIGH BYTE AT 3
WORD	XXX00	LOW WORD AT 0
WORD	XXX01	MID WORD AT 1
WORD	XXX10	HIGH WORD AT 2
WORD	XXX11	LOW BYTE AT 4, THEN HIGH BYTE AT 3
DWORD	XXX00	DOUBLEWORD AT 0
DWORD	XXX01	LOW BYTE AT 4, THEN HIGH 3 BYTES AT 1
DWORD	XXX10	LOW WORD AT 4, THEN HIGH WORD AT 2
DWORD	XXX11	LOW 3 BYTES AT 4, THEN HIGH BYTE AT 3
80386SX:		
BYTE	XXX00	LOW BYTE AT 0
BYTE	XXX01	HIGH BYTE AT 1
BYTE	XXX10	LOW BYTE AT 2
BYTE	XXX11	HIGH BYTE AT 3
WORD	XXX00	WORD AT 0
WORD	XXX01	HIGH BYTE AT 1, THEN LOW BYTE AT 2
WORD	XXX10	WORD AT 2
WORD	XXX11	LOW BYTE AT 4, THEN HIGH BYTE AT 3
DWORD	XXX00	WORD AT 0, THEN WORD AT 2
DWORD	XXX01	LOW BYTE AT 4, HIGH BYTE AT 1, WORD AT 2
DWORD	XXX10	WORD AT 4, THEN WORD AT 2
DWORD	XXX11	WORD AT 4, LOW BYTE AT 6, HIGH BYTE AT 3

Table 5-2. CPU Events for Various Access Widths

In an 80386DX or 80486DX system, memory can be accessed as a byte, word, three-byte, or doubleword. How Periscope shows the memory access in the trace buffer depends on the access width and the starting address modulo 4. All memory accesses except the four cases shown in Table 5-2

are completed in one CPU event. Again, one CPU event may require multiple bus cycles.

Tables 5-1 and 5-2 show the data access types and the CPU events required for various access widths and addresses on the 80386DX, 80386SX, and 80486DX (DX2, DX4) CPUs.

Note these important points about Table 5-2:

- 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 you have BS-16 memory in your system. See the file NOTES.TXT on the Periscope disk for more information.
- On an 80386SX system, you can set data breakpoints on bytes and words when the address ends with 00b or 01b, but not when it ends with 10b or 11b. You cannot set data breakpoints on doublewords, since they will be broken into a minimum of two word accesses. The possible data breakpoints are on low byte, high byte, and word. In short, the 80386SX acts like an 80386DX with all of its memory configured as BS-16 (see above).
- 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 or 80486 system, OUTs to ports 320h through 323h show on the low, mid, next, and high bytes respectively. On an 80386SX system, the above OUTs show on the low, high, low, and high bytes respectively.
- Since Model IV sees everything from the perspective of the CPU, the code, not the physical width (8, 16, or 32 bits) of the memory device, dictates memory moves.
- Since some memory accesses may be split into multiple CPU events, it is important to take the address into account when you set data breakpoints.
- When you use `H3` and `L3` data breakpoints, specify three bytes of data from high to low addresses (to match the trace buffer display).
- To watch for writes of a character to the display, set the memory breakpoint 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.



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 or 80486 system. This technique does not work on an 80386SX system.

```
HM B800:0000 L2 W
HD LW 0750
HD LW 0751
GH
```

The above commands activate Periscope when a low word write of 0750 or 0751 occurs at B800:0000.

5.2.7 Command: Hardware Memory breakpoint (HM)

Syntax:

```
HM [<address> <address> A|H|R|W|X
[B][(s,t)]] [?] [*] [+] [-] [...]
```

Description:

Use this command to set, display (?), clear (*), enable (+), or disable (-) memory breakpoints.

You can enter the breakpoints as two addresses or a range, and you can specify up to eight breakpoints. Indicate the type of operation you want to break on with A, H, R, W or X, for interrupt Acknowledge, CPU Halt, memory Read, memory Write, and code prefetch (instruction eXecution) respectively.



You can substitute Lx (a length in bytes) for the second address in the syntax.

Use the optional qualifier B to cause Periscope to capture breakpoint (trigger) events in the trace buffer but NOT to stop your program's execution. Periscope automatically enables selective capture when you use this parameter. Be sure to set another breakpoint to stop your program's execution.

Use the optional argument (s,t) to set sequential triggers. s is the start state, from 0 to 6, and t is the state 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 access breakpoints inside Periscope's code/data area. Also, watch out for breakpoint boundaries. A memory access breakpoint set at B800:1 won't be triggered by a word or doubleword access at B800:0.

Code Prefetch. The code prefetch breakpoint occurs when memory is read into the prefetch queue. Since the length of the prefetch queue varies from six bytes to 32 bytes, depending on the CPU, instructions are read before they're actually executed. However, due to the 'breakpoint overrun' phenomenon, a breakpoint does not occur immediately. So the instruction causing a breakpoint may or may not have been executed by the time you see Periscope's screen. If the instruction has not been executed, try setting the execution breakpoint after an instruction that flushes the prefetch queue (JMP, CALL, RET, IRET, INT, etc.).

In an 80386 or 80486 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 80486 chip uses burst mode to fetch instructions and read memory quickly in 16-byte (paragraph) chunks. Normally, when the CPU reads a memory location, both the address and the data appear at the pins of the CPU. In burst mode, the address and four double words of data appear at the pins of the CPU, but the addresses corresponding to the last three dwords of data do not appear. Periscope deduces these missing 'burst' addresses from the defined rule set for the address sequence (see the Intel manuals for more information), so it can display and disassemble them in the trace buffer. But Periscope cannot support hardware breakpoints on the missing 'burst' addresses since they never appear at the pins of the CPU.

Use the standard segmented address format to set breakpoints in the first megabyte. To set breakpoints beyond 1MB, up to 16MB, use absolute physical addresses of four digits, an ampersand (&), and the remaining four digits.

Sequential Triggers. The sequential trigger capability is quite powerful and best explained with the following example.

To watch for the writing of the variable BAR only when the routine FOO is executing, enter these commands:

```
HM FOO_START L1 X (0,1)
HM FOO_END L1 X (1,0)
HM BAR L2 W (1,!)
GH
```

When you arm the Model IV board with a `GH` or `GM` command, it always starts in state 0. The first breakpoint advances from state 0 to state 1 on entry to FOO at FOO_START. The second breakpoint reverts from state 1 to state 0 on exit from FOO at FOO_END. The third breakpoint generates a trigger if BAR is written while state 1 is active, i.e., while FOO is executing. If BAR is written while any other state is active, i.e., while FOO is NOT executing, no trigger occurs.

Some restrictions apply:

- The pass count and data breakpoints always apply to state 0.
- Periscope reserves the use of state 7.
- If you specify a true state but do not use it somewhere else as a current state, Periscope displays an error. For example, if you set `(1,5)` as a state setting, you must use the true state 5 as the current state somewhere else, i.e., `(5,x)`.
- Be careful when you use fetches to switch states. CPU pipelining (use of the prefetch queue) can cause fetches to occur earlier than the memory activity performed by the instructions.
- You cannot use the same trigger in two different states.
- If you set `HC TT`, Periscope reserves the use of states 6 and 7.

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 not likely to occur, 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,!)` 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 advances to 1, which enables the memory breakpoint on writes to 0:46C for a length of 2. When this breakpoint occurs, Periscope displays its screen.

`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.

5.2.8 Command: Hardware Port breakpoint (HP)

Syntax:

```
HP [<port> [<port>] I|O [B] [(s,t)]] [?]
[*] [+] [-] [...]
```

Description:

Use this command to set, display (?), clear (*), enable (+), or disable (-) port breakpoints.

You can enter the breakpoints as a single port or as a range of two ports, where each value must be from 0 to FFFFh, and you may specify up to eight breakpoints. Indicate the type of operation you want to break on with an I for In (port read) or an O for Out (port write). Do not attempt to set port breakpoints inside Periscope's port range.

Use the optional qualifier B to cause Periscope to capture breakpoint events in the trace buffer and NOT stop execution. Periscope automatically enables selective capture when you use this parameter. Be sure to set another breakpoint to stop your program's execution.

Use the optional argument (s,t) to set sequential triggers. s is the start state, from 0 to 6, and t is the state moved to when the breakpoint condition is true, from 0 to 6 or ! to cause a trigger. See Section 5.2.7 above for more details.

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.

5.2.9 Command: display Hardware trace buffer (Hx)

Syntax:

`Hx [*] [$] [!] [+] [#<number>] [<file>]`
where `x` is `R`, `S`, `T`, or `U`

Description:

Use this command to clear the hardware trace buffer (`*`); to display the entire buffer from top to bottom (`!`); to force the display of the full buffer, even when it's 'empty'(+); to display the buffer starting at a specific sequence number (`#<number>`); to view the buffer starting where you were the last time you viewed it (`$`); and to display a trace buffer file (`<file>`) you previously created with the `HW` command.

When you use this command without an `*` or `!`, you'll enter an interactive, full-screen display mode, unless the buffer is empty. To exit this mode, press the **Esc** key.

When you use the `<file>` parameter, you must position it as the last parameter on the command line.

Display Formats R, S, T, and U

You can display the hardware trace buffer in four formats:

- **Raw Mode**, which shows all CPU events "as is"
- **Single-entry Mode**, which shows only the breakpoint (trigger) event in Raw Mode format
- **Trace Mode**, which shows all CPU events, but disassembles fetch operations into their instructions
- **Unassembly Mode**, which shows just disassembled instructions

Details on each of the formats follows.

Raw Mode. Use `HR` to display the real-time trace buffer in a 'raw dump' format .

Each line of the display corresponds to one CPU event and 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 (in brackets), a sequence number, and possible other information.

See Figure 5-2 for a sample display. See also the section below titled "Layout of the Trace Buffer Display" for a detailed description of this format.

```
15C4:0138 A100 17E8 Fetch START                    [0B] -07B
15C4:013C 10BF 0134 Fetch                          [02] -07A
15C4:0140 0024 E801 Fetch                          [02] -079
15C4:FFFC      013B Write                          [02] -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 CBBC 0134 Fetch                          [02] -074
15C4:0160 C32B EBD3 Fetch SAMPLE#52                [02] -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] -060
15C4:FFFC      013B Read                           [02] -06C
```

Figure 5-2. Hardware Trace Buffer in Raw Format

Single-entry Mode. Use `HS` (with no parameters) to display the single CPU event that caused a hardware breakpoint.

The display output is identical to the raw mode format described above.

Trace Mode. Use `HT` to display the hardware trace buffer in a disassembly-and-data format .

Periscope disassembles code prefetches into one or more lines. The disassembled lines show an address, the opcodes

making up the instruction, and the instruction itself, and/or the source code line. Periscope decodes the address by the rules described below under "Layout of the Trace Buffer Display". If an instruction matches a source line and DOS is available, Periscope displays the source line. See the section below titled "Analyzing and Examining the Trace Buffer" for a discussion of Periscope's trace buffer analysis.

Periscope displays other CPU operations (read, write, in, out, etc.) in the raw mode format described above.

See Figure 5-3 for a sample display.

```
   #29: start:  call getmem          ; get memory size
15C4:FFFC       013B  Write,                          [13] -078
   #44:         mov cl,6             ; shift count
   #45:         mov si,2             ; point to top of memory in psp
   #46:         mov ax,[si]          ; get to 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       013B  Read                         [02] -06C
```

Figure 5-3. Hardware Trace Buffer in Trace Format

Unassembly Mode. Use **HU** to display the hardware trace buffer in a disassembly-only format .

Periscope disassembles code prefetches in the trace mode format described above. It does not display any other CPU operations (read, write, in, out, etc.).

See Figure 5-4 for a sample display.

```
#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 5-4. Hardware Trace Buffer in Unassembly Format

Layout of the Trace Buffer Display

For the trace and unassembly mode formats (see Figures 5-3 and 5-4), Periscope disassembles prefetch operations and displays an address, opcodes, and an instruction on each line. You'll see symbols and/or source whenever possible. In trace mode it displays CPU operations other than prefetches in the raw mode format described below.

The raw mode format of the trace buffer display (see Figure 5-2) contains six fields:

- Address
- Data
- Operation
- Cycle count
- Sequence number
- Periscope ID (when applicable)

Address Field. Periscope displays the address in one of four formats:

- a segmented address (the segment, a colon, and the offset), or
- a four-digit port number, or
- a five-digit non-segmented address, or
- an eight-digit absolute physical address (four digits, an ampersand, and four digits)

Periscope displays the *segmented address* for code and data references when it can decode the address into one of the following segments:

- User-specified segment (use the `S <segment>` command while viewing the trace buffer)

- A000, B000, C000, D000, E000, or F000 (BIOS segments)
- RUN's last PSP segment
- Current derived segment calculated from RETF and IRET instructions
- Periscope's segment
- DOS's segment

Periscope displays the *five-digit non-segmented address* when it is unable to decode the segment for memory in the first megabyte into one of the above values. It uses the *four-digit port number* for I/O port access. It displays an *eight-digit absolute address* for addresses above one megabyte.

Data Field. Depending on the CPU you're using and its method of memory access, the data field may be a byte, word, three-byte, or doubleword. See Tables 5-1 and 5-2 for more information on the possible data types. The data is up to four bytes, from high to low, in fixed columns. The address is for the lowest (rightmost) byte of the data.

Operation Field. The possible operations 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 by `HC C+`)
- Write (memory write)
- Void (invalid trace buffer record)

All operations except `Probe` are mutually exclusive. A symbol name may follow the operation if the address indicates a symbol. If the operation uses an interrupt vector as its address, `INT XX` will follow the operation. If the operation is a read, write, in or out of a byte value in the ASCII range, the ASCII character appears in quotes after the operation. On 80486 systems, the word `Burst` follows the `Fetch` or `Read` operation for burst reads.

Cycle Count Field . The cycle count is the total number of CPU cycles that have occurred since the previous trace buffer operation. Periscope shows the cycle count as two, four, or five hex digits in brackets. If it finds an illegal value, it displays `?` after the value. (If you see an illegal value, please call Technical Support.) You may see a plus sign following the cycle count on overflow records. This legitimately happens when too many CPU cycles occur between trace buffer records and the cycle count capture control is disabled (`HC C-`).

The 8-bit cycle count field holds up to 127 cycles per trace buffer record, enough for most continuously captured instructions. If you're using selective capture and want to count the CPU cycles between events, turn cycle capture on with `HC C+` to force the capturing of cycle records. The resulting records have an operation type of `Cycle` and are displayed only in `HR` mode.

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 enter `#` and a sequence number or a search command to move to the second item. (See the section below titled Trace Buffer Commands.) You'll see the accumulated cycle count in the cycle count field in hex.

In `HT` mode, Periscope does not display cycle records. It accumulates cycle counts and displays them on the next non-cycle record.

To convert CPU cycle counts to time, use Table 5-3.

CPU SPEED (MHz)	TIME PER CYCLE (ns)
8	125
10	100
12.5	80
16	62.5
20	50
25	40
33	30
40	25
50	20
66	15

Table 5-3. Cycle Time by CPU Speed

Sequence Number Field. The sequence number (in hex) may be from -3FFE to +3FFE, depending on the trigger location. (See Table 5-4 for specific sequence number ranges for the various trigger locations.) Notice that the center trigger captures half the buffer length after the trigger event before stopping.

TRACE	BOTTOM TRIGGER	CENTER TRIGGER	TOP TRIGGER
16K	-3FFE TO 0	-1FFF TO 0 TO +1FFF	0 TO 3FFE

Table 5-4. Trace Buffer Sequence Ranges

`Top` displays after the sequence number of the first entry in the buffer. `End` or `Bottom` displays after the sequence number of the last entry in the buffer. A sequence number of zero indicates the trigger (breakpoint) event.

Periscope ID Field. If the address is within Periscope's code/data area, Periscope displays `PS` to identify the code/data as its own. You may see these `PS` entries at the beginning and end of the trace buffer. They show where Periscope turned control over to the application and regained control from the application.

Key Usage. When you display the trace buffer in the full-screen, interactive mode (not using * or !), you can use the **Home**, **End**, **Up**, **Dn**, **PgUp**, **PgDn**, **Left Arrow**, **Shift-Up** and **Shift-Down** keys to move around in the buffer.

Press the **Home** key to get to the top of the buffer. Press the **End** key to get to the bottom of the buffer.

Use the **Up** and **Dn** keys to move up or down by one line. Use the **PgUp** key to move up one screen and the **PgDn** key to move down one screen.

Press the **Left Arrow** key to get to the center of the buffer (actually, 8K from the end of the buffer). Use **Shift-Up** and **Shift-Down** to move up and down in the buffer by 80h (128) records.

Press **Esc** to exit.

Flushing unexecuted instructions. The first time you view the trace buffer in trace (HT) or unassembly (HU) mode format, Periscope analyses the entire trace buffer to determine what code was actually executed. This analysis can take up to a minute or more for a complex 16K trace buffer. You can view the buffer while the analysis is taking place, but movement in the buffer may be slower than normal and the display may change once the analysis is complete. Periscope does not accumulate CPU cycle counts while the analysis is running.

Even though code has been fetched by the processor, there is no absolute method of telling whether the code was actually executed. Periscope attempts to infer the execution of code in the buffer during the analysis described above. When it finds an unconditional JMP, CALL, RET, RETF, INT, or IRET instruction, it "flushes" the buffer since it knows the instructions in the buffer after the unconditional instruction were not executed. To improve readability, Periscope displays a blank line after any instruction that causes a change in program flow.

Periscope cannot always correctly determine whether a conditional jump was taken, nor can it always determine the correct execution at the top and bottom of the buffer since it may need information outside the buffer to make the determination. When you see the message `Skip=x` on the right side of the screen, you know that Periscope is not sure it has made the correct determination. It displays the message to remind you that you can force the desired start byte with the trace buffer commands `0`, `1`, `2`, `3`, and `X` described below under Trace Buffer Commands. You can use the Skip command to correctly display any instructions that Periscope flushed because it incorrectly determined that they were not executed.

You'll see memory and port accesses (read, write, in, or out) one or more lines after the instruction that performed them. These memory and port accesses may be the only way you can determine for sure that an instruction was executed.

Use `HA *` to clear the trace buffer analysis information and force Periscope to perform the analysis again the next time you use `HT` or `HU`.

Each time you display a saved trace buffer file (using the `<file>` parameter), Periscope analyzes it. Use the `$` parameter to keep Periscope from re-analyzing a file it has previously analyzed. This works only if you have not used a `G`, `J`, or `T` command since Periscope analyzed the file.

486 Burst Mode. The word `Burst` may follow `Fetch` and `Read` operations on an 80486 system when Periscope can deduce the burst addresses. When Periscope cannot determine burst addresses because the base record it needs is not present in the trace buffer, it instead displays `Void, Burst` in `HR` mode. It does not display these entries in `HT` and `HU` modes. See Section 5.2.7 on page 61 for more information.

Register contents. Register information is not explicitly available in the hardware trace buffer, but you can deduce register values. Look 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.

COMMAND	DESCRIPTION
/A <address>|<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 byte for first instruction
2	Skip 2 bytes for first instruction
3	Skip 3 bytes for first instruction
X	Restore skip state to initial value
"	Toggle duplicate display (source & assembly)
*	Toggle flushed prefetch display
C	Toggle cycle count display
E	Toggle extended memory display
F <address> <address>	Fixup address
S	Add Segment to list
S-	Remove Segment from list
S+	Enable Segment decoding
S-	Disable Segment decoding
# <number>	Move to record
J <number>	Jump to bookmark
P <number>	Place a bookmark
A	Use Asm mode
B	Use Both mode
R	Use HR display
T	Use HT display
U	Use HU display

Table 5-5. Summary of Model IV Trace Buffer Commands

Trace Buffer Commands. While you are using the HR, HT, or HU commands, you can enter ? to display help on the functions available, or press **Alt-M** or **F10** to activate the menu system. Use the trace buffer commands shown in Table 5-5 and described below to control the display of and to search through the trace buffer.

- To ***switch from one buffer display mode to another***, press `R` (Raw), `T` (Trace), or `U` (Unassembly). To switch to Assembly-only mode (like the `UA` command), press `A`. To switch to Both mode (like the `UB` command), press `B`. To toggle between source-only and mixed (source and assembly) modes, press the double quote (`"`). The default is mixed mode.
- To ***move to a specific location in the buffer***, enter # followed by the sequence number. The sequence number must be from `-3FFE` to `+3FFE`. Any value outside this range is ANDed with 3FFFh.
- To ***move to a previously marked location***, enter `J` followed by a 'bookmark' number. The bookmark number can be from `0` to `9`. You assign bookmarks (place markers) on locations you may want to return to, then use the `P` command to return to them.
- To ***toggle the CPU cycle count display on and off (default) for disassembled lines***, press the letter `C`. When you use this mode, 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.
- To ***toggle the display of extended memory on (default) and off***, press the letter `E`. When the toggle is off, Periscope does not display memory beyond one megabyte (an address of 0010&0000 or higher). You may want to toggle `E` off if you're using your Model IV with a memory manager such as 386MAX or QEMM. This will suppress CPU events for the memory manager's access to extended memory, which occurs frequently.
- To ***toggle the display of flushed instructions on and off (default in*** `HT` ***or*** `HU` ***mode)***, enter an asterisk (`*`). When you enter an asterisk, Periscope displays flushed instructions preceded by an asterisk, and it displays an asterisk instead of the usual colon after the line number.
- When it analyzes the trace buffer, Periscope deduces the segments used by the instruction stream both for code and data. It saves those segments in a list that it uses for decoding when you display the buffer. To ***add a segment to Periscope's segment list***, enter `S <segment>`. To

remove a segment from the list, enter `S-<segment>`. To ***disable segment decoding*** so that no segments are displayed, enter `S-`. To ***enable segment decoding***, enter `S+`.

- 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 tell Periscope to skip that number of bytes for the first instruction only. (Only `0` and `1` are valid on an 80386SX CPU.) 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 may not correctly track JMPs to registers (e.g. JMP AX), LOOPs or conditional JMPs. To ***return to the starting state or to mark an instruction as executed when Periscope thinks it was not executed***, enter `X`.
- To ***"fixup" an address***, enter `F <address> <address>`, where the first address is the address that is to be replaced and the second address is the address that the first address is to be replaced by. The second address must be less than the first address. 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. You can use this command to fixup addresses beyond one megabyte, which Periscope displays as 8-digit absolute addresses, too. For example, entering `F 12&3456 FOO` fixes up 12&3456 to the address of the symbol FOO.
- 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, remember that a word access to B000:0 would have accessed B000:1, but would 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***, enter the absolute form of the address, `xxxx&xxxx`.

- To ***search for data values on an 80386 or 80486 system***, enter `/D <byte> <byte> <byte> <byte>`. For wildcard bytes enter a question mark. ***On an 80386SX system***, enter just the first two byte fields.
- To ***filter for (display only) records of a single operation 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. You cannot filter on cycle count or void records. Using the type search (below) 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. You cannot search for cycle count or void records.

Examples:

`HR` displays the last page of the trace buffer in raw mode.

`HR *` clears the hardware trace buffer.

`HR PSBUF.DAT` reads the file PSBUF.DAT for the trace buffer information.

`GH;HS` enables hardware breakpoints and displays the breakpoint event from the hardware trace buffer after a hardware breakpoint occurs.

`HT` displays the last page of the hardware trace buffer in mixed mode.

`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 you used `HT`.

Ctrl-P and `HU ! #-200` dumps the trace buffer starting at sequence number -200 to the printer.

5.2.10 Command: Hardware Write (HW)

Syntax:

`HW <file>` or `HW A:` or `HW B`

Description

Use this command to save the contents of the hardware trace buffer to a disk file or to a floppy disk. You can then view the saved trace buffer with the `HR`, `HT`, and `HU` commands.

When you enter `HW <file>`, Periscope uses DOS to write the `<file>`. Table 5-6 shows the file size and track usage for the 16K trace buffer.

TRACE BUFFER SIZE	FILE SIZE	TRACKS USED
16K	163,850	27-37 on head 0 and 8-37 on head 1

Table 5-6. Trace Buffer File Size and Track Usage

`HW A:` and `HW B:` use BIOS calls only (INT 13h) and therefore do not require DOS to be available. Before you enter this command, place a blank formatted floppy disk in the specified drive (`A:` or `B:`). Periscope uses the tracks shown in Table 5-6. This method does not use the DOS directory, so it may partially or totally overwrite any existing files! To create a DOS-readable buffer file (PSBUF.DAT) from the floppy disk file, run PS4TEST with `/WA` or `/WB` options after you return to the DOS prompt.

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 PSBUF.DAT`

If DOS is busy, place a formatted disk in drive A and enter `HW A:`. Then get to the DOS prompt and enter `PS4TEST`

/WA. Get back into Periscope and enter `HT PSBUF.DAT` to display the trace buffer.

5.2.11 Command: toggle internal 486 cache (/4)

Syntax:

`/4`

Description:

Use this command to enable or disable the cache that is an integral part of the 80486 chip. When the internal 486 cache is enabled, the hardware trace buffer display is generally unintelligible because there may not be any corresponding memory reads or fetches at the pins of the CPU where Periscope can see them. This is not a problem on 80386 systems since all cache on these systems is external to the CPU.

Examples:

`/4` changes the state of the internal 486 cache, i.e., if it was enabled, it will be disabled and vice versa.

CHAPTER SIX

Tips on Using Model IV

- Tracking Program Flow
- Examining the Trace Buffer After a Crash
- Debugging the Power-on Startup Tests (POST)
- Capturing Specific Code
- Detecting Hardware Interrupts
- General Tips
- Hardware Breakpoint Examples

This chapter gives you some practical information on using Model IV to debug in various situations.

6.1 TRACKING PROGRAM FLOW

If you're debugging a complex program and need to get a high-level overview of the program's flow, embed writes of unique values that indicate where you are in the program to a dummy memory location or I/O port. Then set the Model IV board to selectively capture writes to the memory or I/O port that you've used.

Using this technique, you can more readily track the flow of the program. Also, if you want to set a breakpoint on the execution of a particular location, you can set a hardware breakpoint on the write of the memory or port, qualified by the unique data value written at that location.

For example, assume we're using a dummy I/O port of F310 and outputting values ranging from 0 to 3F at various points in the program. To capture the next 100h outs to that port and nothing else, use the following commands:

```
HA *
HC #100 S+
HP F310 O
GH
```

To capture the location where the value 30h is being written to the port in the center of the trace buffer, use the following commands:

```
HA *
HC TC
HP F310 O
HD LB 30 30
GH
```

6.2 EXAMINING THE TRACE BUFFER AFTER A CRASH

If you're debugging a program that crashes the system, you'll find that in many cases an exception interrupt will activate Periscope. From there, you can look in the real-time trace buffer to see what lead up to the exception.

In some cases, the system may have executed meaningless, but legal instructions for the full depth of the buffer, leaving you with nothing to indicate where things went awry. If this happens, try filling unused memory with FF's which will cause an exception when executed. Or try setting a hardware breakpoint on an event you see near the top of the trace buffer.

If the system is so far gone that Periscope can't come up, all is not lost. Install the Model IV on a system that has a reset switch. Then after the crash, do the following:

- Press the break-out switch to stop the trace buffer (it will also stop if it reaches a hardware breakpoint).
- Press the reset switch to reboot the system without turning the power off.
- While the system is booting, press the **Alt** and **Ctrl** keys to keep Periscope from reloading and destroying the state of the trace buffer.
- Run `PS4TEST/W` to save the trace buffer to the file PSBUF.DAT.
- Load Periscope normally and use the `HT PSBUF.DAT` command to view the saved trace buffer.

6.3 DEBUGGING THE POWER-ON STARTUP TESTS (POST)

To debug the POST sequence, set Model IV up for passive remote mode and then do the following:

- Boot both host and target systems, then load Periscope on the host.
- Enter `HC TT` to force the trigger (breakpoint) event to the top of the trace buffer.
- Enter `HM FFFF:0 L1 X;GH` to set a breakpoint on a fetch of the last paragraph in the first megabyte, then to go with breakpoints set.
- When the breakpoint occurs, do a long boot on the target system using Periscope's `QL` command or equivalent.

6.4 CAPTURING SPECIFIC CODE

In many situations, you may want to capture everything that occurs from one point in your program to another, but not

capture anything that occurs as a result of any other code execution. The easiest way to do this is to embed a unique event at the beginning and end of the code stream that you want to monitor. Use of a dummy I/O port read event is both easily added to the code and guaranteed (assuming you choose a good I/O port) to be unique. Note that use of a code fetch event may not be an ideal choice because of the prefetch queue.

You can set up hardware breakpoints to change from state 0 to state 1 at the beginning of the code stream and change from state 1 to state 0 at the end of the code stream. Then set the hardware controls so that nothing is captured in the hardware trace buffer while in state 0 and issue the `GH` command to start execution. You'll have to manually stop Periscope, using the break-out switch or hotkeys, but nothing except the desired code stream (and its I/O and memory activity) will be captured in the trace buffer.

For example, assume you want to capture execution from point A to point B in your code and that you have an In from port 1234h at point A and an In from port 1235h at point B. You'd enter the following commands to capture just the execution of code from point A to point B:

```
HP 1234 I (0,1)
HP 1235 I (1,0)
HC X0
GH
```

6.5 DETECTING HARDWARE INTERRUPTS

When a hardware interrupt occurs, it is preceded by two interrupt acknowledge cycles. On 80386 and higher systems, the second cycle indicates the hardware interrupt number in the low byte. You can use it to selectively capture hardware interrupts, which will give you a high-level overview of the interrupt sequence in your system. To set a breakpoint on this interrupt acknowledge cycle use the following command:

```
HM 0:0 L1 A
```

Note that the address of the first Int Ack cycle is 0:4 and the second, useful one is 0:0.

6.6 GENERAL TIPS

Bus Compatibility. Since Model 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 (direct memory access) signals so you cannot set breakpoints on DMA events or capture them in the trace buffer. This means that Model IV cannot detect memory modified by DMA.

Data Breakpoints. Setting data breakpoints can be tricky, since Model IV sees everything from the perspective of the CPU. See Section 5.2.6 on page 56 for more information.

Breakpoint Overrun. A phenomenon called breakpoint overrun occurs with all hardware breakpoint devices. Between the time a breakpoint occurs and the time NMI stops the processor, the processor executes one or more instructions. When you view the trace buffer with the `HR` or `HS` command, the CPU event shown with a sequence number of zero 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 (which do not show up in the trace buffer) will be executed after 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/EM's `BM` command for this job.)

On 80386 and 80486 systems, the breakpoint overrun may be up to six instructions, although it is usually two to four instructions.

Rebooting. Don't reboot the system using **Ctrl-Alt-Del** when the board is armed (i.e., when you've entered a `GH` or a `GM` command). If a breakpoint occurs during the boot, your system may hang. Press the break-out switch to go into Periscope and use any exit other than `GH` or `GM` to disarm the board.

386 and 486 Prefetch. On 80386 and 80486 systems, 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, Model IV adjusts the address to the next lower doubleword boundary.

Different CPUs have different prefetch queue lengths. In our testing, we've seen 8-, 12-, 16-, and 32-byte prefetch queues. To determine the length of the prefetch queue on your system, enter `RUN` with no arguments, then use the in-line assembler to assemble a `JMP 100` instruction. Let this infinite loop execute for a second, then press the break-out switch to get into Periscope. Now enter `HR` to see the trace buffer in raw mode format. Count the number of bytes that are fetched starting at CS:100 to get the minimum size of the prefetch queue. Instructions that do more work, such as multiply, can create a longer prefetch queue.

6.7 HARDWARE BREAKPOINT EXAMPLES

- To break in BIOS, enter `HM * F000:xxxx L1;GH` where `xxxx` is the desired IP in ROM. This traps on a prefetch of the specified location, usually stopping when the instruction is about to be executed. On an 80386 or higher system, you can also use Periscope's `BD` command to take advantage of the 80386 debug registers.
- To trap set cursor calls, enter `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 equals one, Periscope displays its screen, otherwise it resumes full-speed execution.
- To trap writes to interrupt vectors by your program, enter `HM 0:0 0:3FF W;BR CS EQ CS;GM`.
- If you suspect your program is underflowing its stack, enter `HM SS:0 L3 W;GH` to trap writes to the bottom of the stack, assuming that the lower boundary of the stack is at SS:0.
- To monitor interrupt vector reads, enter `HM * 0:0 3FF R` and then `GH;HS`.
- To set a hardware breakpoint on 80386 shutdown, enter `HM 0:0 L1 H`. To set a breakpoint on a halt, enter `HM 0:2 L1 H`.

- To trap an overrun error on COM1, watch for a read of port 3FDh with bit 1 on. The Model IV 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.

APPENDIX A

Model IV Messages

You'll find details on the error messages specific to Model IV in this chapter. (Most are generated by the diagnostics program, PS4TEST.) See the Periscope manual for all other messages displayed by Periscope/EM and Periscope/32. See the PopUp Periscope manual for all messages displayed by PopUp Periscope.

39—Invalid HM/HP settings

An invalid sequential trigger state was used with the `HM` or `HP` command. Valid state numbers are from `0` to `6` (when you use `HC TT`, the upper limit is `5`). Also, any true state must be used as a current state.

67—Unable to read trace buffer -- run PS4TEST

Periscope was unable to read the hardware trace buffer. Check the items listed under the description of PS4TEST in Section 3.3.10 on page 31 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

76—Periscope Model IV board not found

Check the installation of the Model IV board. It is likely that the base I/O port is not set to the default of 300h or that the port specified with the `/P:nnn` installation option does not match the port used by the board.

400 through 408

These errors indicate a potentially serious diagnostic failure. Check the items listed under the description of PS4TEST in Section 3.3.10 on page 31 and try again. If errors persist, call Technical Support.

409—Invalid option

An invalid command-line option was used. Enter `PS4TEST ?` to display the valid options.

410—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.

411—Unable to read diskette

PS4TEST was unable to read the saved trace buffer from a floppy disk. Check the disk and try again.

413—Break-out switch failed

The break-out switch test failed. Check the items listed under the description of PS4TEST in Section 3.3.10 on page 31 and try again. If errors persist, call Technical Support.

414—Periscope Model IV board not found

Check the installation of the Model IV board. It is likely that the base I/O port is not set to the default of 300h or that the port specified with the `/P:nnn` installation option does not match the port used by the board.

Glossary

The terms in this glossary apply to hardware-assisted debugging in general and the Periscope Model IV hardware in particular. If you encounter terms in this manual or in the PopUp Periscope manual that are not defined here and that you do not understand, please call Technical Support for assistance.

386 or 80386DX

The Intel x86 CPU introduced after the 286. This CPU runs at speeds from 16MHz to 33MHz.

386EX or 80386EX

The 386EX is a highly-integrated version of the 386SX, with many features that previously required external circuitry built into the chip. Although it can operate at voltages other than 5V, Model IV's support (in the *Embedded Periscope* products) is currently limited to 5V.

386SX or 80386SX

This CPU is similar to the 386, but has a 16-bit (instead of a 32-bit) data path. It is popular in lower-cost 386 systems. Model IV supports this chip at speeds up to 25MHz.

386EX Adapter or 80386EX Adapter

This Model IV adapter converts the 132-pin Plastic Quad Flat Pack (PQFP) 80386EX chip, which is typically surface-mounted, into an 80386DX Pin Grid Array (PGA) connector so that you can use a 386 pod. This adapter is part of the *Embedded Periscope* product line.

386SX Adapter or 80386SX Adapter

This Model IV adapter converts the 100-pin Plastic Quad Flat Pack (PQFP) 80386SX chip, which is typically surface-mounted, into an 80386DX Pin Grid Array (PGA) connector so that you can use a 386 pod.

486 or 80486DX

The Intel x86 CPU introduced after the 386. This CPU runs at speeds of 25, 33 and 50MHz. Model IV supports the chip at 25MHz and 33MHz, but does not support it at 50MHz.

486DX2 or 80486DX2

This CPU runs at speeds of 25MHz or 33MHz externally, and at speeds of 50MHz or 66MHz internally. Model IV supports this chip, since the external speed (at the pins of the CPU) is the speed that matters with Model IV.

486DX4 or 80486DX4

This CPU runs at speeds of 25MHz or 33MHz externally, and at speeds of 75MHz or 100MHz internally. Model IV supports this chip, since the external speed (at the pins of the CPU) is the speed that matters with Model IV.

486SX or 80486SX

This CPU is like a 486DX with a numeric coprocessor built in. The Model IV hardware supports the PGA version of this chip at external speeds up to 33MHz.

486SX2 or 80486SX2

This CPU is a 486SX which runs at clock-doubled speeds internally. The Model IV hardware supports the PGA version of this chip at external speeds up to 33MHz.

Active Remote Mode

To run in active remote mode, install the Periscope Model IV board and Periscope/EM or Periscope/32 debugger software on the host system, install a Periscope Model IV pod and Periscope remote software (for DOS, Windows, OS/2, etc.) on the target system (where the program you want to debug runs), then connect the host and target systems with a null modem cable.

Break-out switch

The push-button switch that is included with all Periscope debuggers. It generates the non-maskable interrupt (NMI, aka INT 2) signal, which activates the debugger software. Use the break-out switch to recover from system crashes or to simply

stop your program and enter the debugger.

Breakpoint

The point in a program where you wish to stop and examine the state of the system. Think of it as "take a Break when you get to this Point". There are various types of breakpoints, some implemented in software, others in hardware. See also Hardware Breakpoints and Software Breakpoints.

Breakpoint Overrun

This term refers to the instructions that are executed during the time it takes the CPU to stop after a hardware breakpoint occurs. On 386 and later systems, the breakpoint overrun is usually two to four instructions.

BS-16 Memory

The 386 and later CPUs can be connected to both 16-bit and 32-bit data busses. For performance reasons, most systems use 32-bit memory for all address ranges. However, some systems use 16-bit memory in the ROM region (A000:0 to F000:FFFF), which is referred to as BS-16 memory. 386SX systems contain only BS-16 memory. On these systems, setting data breakpoints with Model IV can be difficult, due to the remapping of data values.

Burst Mode

The 486 CPU uses burst mode to fetch instructions and read memory quickly in 16-byte (paragraph) chunks. In burst mode, not all addresses actually appear at the pins of the CPU; the missing addresses must be deduced by following an Intel-defined set of rules. Instead of the normal address then data sequence four times, burst mode returns an address and then four data values. The addresses of the last three data values must be deduced from the starting address.

Cache

The 486 CPU has an internal cache that you can enable or

disable. When you enable the internal cache, the Model IV hardware trace buffer display is not generally meaningful because not all information is available to Model IV. This problem is solved when you disable the internal 486 cache.

Caches external to the pins of the CPU are never a problem, since Model IV watches the system from the pins of the CPU. Since 386-based systems have no internal cache, there is never a cache-related problem on these systems.

CPU

An acronym for Central Processing Unit. In the PC family, this refers to the 8088, 80286, 80386, 80486, Pentium, etc., chips.

CPU Event

The most basic CPU activity, such as a memory read, write, or instruction fetch, or a port read or write. By default, each CPU event is captured in the Model IV trace buffer as an 80-bit wide "record" that includes a 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. Periscope Model IV collects these events in real-time (i.e., with no system slowdown) in its hardware trace buffer, which can hold up to 16,384 events.

Cycle Count

The count of CPU clock cycles since the previous trace buffer event. Use this information to calculate the time it took a series of events to execute. For example, on a 33MHz system, each clock cycle is 30 nanoseconds, so you can multiply the total number of cycles shown in the trace buffer for a series of events times 30 nanoseconds to determine the execution time for those events.

See Table 5-3 on page 72 for more information.

Debug Registers

Starting with the Intel 386 processors, the CPU supports four debug registers that allow software debuggers to provide real-time breakpoints on reads, writes, and execution of memory from one to four bytes in length.

Direct Memory Access (DMA)

DMA is memory reads and writes that are performed by a processor other than the CPU. Since Model IV watches the system from the perspective of the pins of the CPU, you cannot use it to set breakpoints on DMA events. Generally, you'll need to use a bus-based device to debug DMA problems.

EISA

This acronym stands for Extended Industry Standard Architecture. It is a 32-bit extension of the original ISA bus.

EM Memory

The write-protected, extended memory above one megabyte that Periscope/EM can run in when you are running it with a supporting memory manager. Periscope/EM uses no low-DOS memory and only 32-36K in high-DOS memory when the EM feature is used.

Halt

This is one of the states of the CPU. It is invoked with a halt (HLT) instruction. Once the Halt state is invoked, program execution stops until a hardware interrupt, NMI, or reset occurs.

Hardware Breakpoints

Breakpoints (see Breakpoint) implemented in hardware that operate in real-time, i.e., without slowing the system down. They include breakpoints on memory ranges and I/O port ranges that can be qualified by data or bit values as well as a

pass counter. These are the kind of breakpoints provided by debugger hardware such as in-circuit emulators, logic analyzers, and hardware-assisted debuggers like Periscope Model IV.

Hardware Trace Buffer

A circular buffer that captures the execution history of the system in real-time. Using the hardware trace buffer, you can see events leading up to (and following) a breakpoint without slowing your system down. This is the real-time buffer provided by debugger hardware such as in-circuit emulators, logic analyzers, and Periscope Model IV. Model IV's hardware trace buffer can hold up to 16,384 CPU events.

See also CPU Event.

Host Debugger

The software debugger that runs in the host system in a remote or embedded environment. For example, Periscope/EM and Periscope/32 can be host debuggers when you run Model IV in active remote mode.

Host System

The development machine or "debugging station" where the host debugger runs.

See also Target.

Interrupt Acknowledge

This is one of the CPU events that occurs when a hardware interrupt begins. For example, when a timer tick begins, two interrupt acknowledge cycles occur. The first one has an address of 0:4 and a byte data value of 0FFh and the second one has an address of 0:0 and a byte data value of 08h (indicating interrupt 8, the timer tick).

ISA

This acronym stands for Industry Standard Architecture. It is the original bus used in the IBM AT that supports 8-bit and 16-bit access, but not the 32-bit access supported by EISA systems.

MCA

An acronym for Micro Channel Architecture. This is the bus in most IBM PS/2 computers. You can use Periscope Model IV in remote mode to debug software running on PS/2 machines.

Memory Manager

A 386 control program, such as 386MAX, QEMM, or NETROOM, that provides high DOS memory, EMS memory, VCPI, and DPMI services. These memory managers also enable Periscope/EM to run from extended memory.

See EM Memory.

Model IV board

The full-length board that provides the hardware trace buffer and hardware breakpoint logic. The board does not function without a separately-purchased processor-specific pod. Use in 80386DX, 80386SX, 80386EX, 80486DX, 80486DX2, 80486DX4, 80486SX (PGA only), and 80486SX2 (PGA only) systems running at external speeds up to 33MHz and internal speeds up to 100MHz.

NMI Clip

A cable assembly that connects the pod to the I/O channel check (NMI) signal on the bus of the target system to stop the CPU when an NMI occurs. You use this clip if you're running Model IV in active remote mode.

Non-Maskable Interrupt (NMI)

This interrupt has the highest priority in the system, and is

thus well-suited for use by debuggers. When you press the Periscope break-out switch, it generates an NMI signal that a software debugger can intercept.

Null-modem Cable

The cable that connects the host system with the target when you run in active remote mode.

Passive Remote Mode

This is a method of debugging with the Periscope Model IV board and Periscope/EM or Periscope/32 software installed in one system (the host) and the Model IV pod installed in another system (the target). No debugger software runs in the target system.
When you run in passive remote mode, your only access to the target system is via the Model IV hardware. This enables you to capture an execution history of the target CPU in the Periscope hardware trace buffer, but you cannot stop the target system, nor can you interactively debug the target. Source level support is generally not available. However, because no software monitor is running in the target, there is no intrusion.

PC-104 Bus

A bus commonly used in embedded systems. It is pin-for-pin compatible with the 16-bit ISA bus. *Embedded Periscope* products support the PC-104 bus.

Pin 1

Pin 1 is used to indicate the correct orientation of the CPU chip. It is indicated by a notched corner and/or by a dimple near one corner of the CPU chip. The silk-screened outline of the CPU on the motherboard may also indicate Pin 1.

Pin Grid Array (PGA)

This is the package used for the Intel 80386DX and 80486DX, DX2, DX4, and some 486SX, and SX2 chips. If you have a PGA chip, no adapter is required to use a Model

IV pod.

Pipelining

See Prefetch Queue.

Plastic Quad Flat Pack (PQFP)

This is the package used for 80386SX and 80386EX chips. If you have a system with one of these CPUs, you will need an adapter in order to use Model IV.

Plus board

This is what the Periscope Model I board is called when you use it as an optional add-on to a Model IV. It provides write-protected memory for the Periscope/EM software, and enables you debug at ROM-scan time. It is generally useful only when you're debugging in a single system DOS environment.

Pod

This is the Model IV CPU interface board. It connects to the CPU and prepares the CPU signals for the Model IV board.

Prefetch Queue

The Intel CPU family reads instructions in clumps. For example, after a call to a function, the processor needs to read the new instructions starting at the function. Depending on the CPU type, it will read up to 32 bytes of instructions before doing anything else. This reading of memory to get instructions is called a "fetch" event. The processor adds the bytes read to its internal prefetch queue so that it can start cracking the instructions for execution.

Probe

This is a status bit shown in the Model IV trace buffer. By default, the probe indicator is shown when a hardware interrupt is pending. By changing a jumper on your pod, you can also have this status bit indicate the state of an external signal

or the Hold Acknowledge signal.

Protected Memory

The memory on the Periscope Model I/Plus board. It is write-protected so that the Periscope/EM software cannot be over-written.

Real-Time Debugging

The ability, only available with debugger hardware, to run a program at full speed while debugging. This ability is critical in the development and debugging of time-critical applications, such as communications software, control programs, and many embedded systems.

Remote Debugging

This is debugging software running on one system (the target) from another system (the host).

See Active Remote Mode, Passive Remote Mode, and Single-System Mode.

Ribbon Cable

The 50-pin cable used to connect the Model IV board and a pod. This cable comes in two lengths: 18 inches for use in a single system, and 48 inches for use in remote mode. The 48-inch cable is shielded to prevent signal degradation at high speeds.

Sequential Breakpoints (Triggers)

This refers to Model IV's state machine capability. Sequential breakpoints allow you to switch breakpoint settings in real-time. For example, if you want to watch for a write to the variable FOO, but only while the function BAR is active, you'd use sequential breakpoints to switch from state 0 to state 1 on entry to BAR and switch from state 1 to state 0 on exit from BAR. Then you'd set a breakpoint on writes to FOO that is activated only when the state is 1.

Single-System Mode

When you run Periscope Model IV with all its hardware and software installed in one system, it is called single-system mode. This is the normal method for using Model IV to debug a real-mode DOS environment.

See Remote Debugging, Active Remote Mode, and Passive Remote Mode.

Software Breakpoints

Breakpoints (see Breakpoint) that are implemented by software-only debuggers. These include temporary or sticky (remembered) code breakpoints, which work by replacing an instruction with a single-byte interrupt 3 (0CCh), and monitor breakpoints that can have a significant impact on system performance. Debug registers available on 386 and later CPUs enable software-only debuggers to provide real-time monitor breakpoints on small ranges of memory.

Software Trace Buffer

The trace buffer software-only debuggers provide, which usually includes a trace of instructions and machine registers. It does not operate in "real-time", because logging a software trace buffer will significantly slow the system down.

See also Hardware Trace Buffer.

State Machine

See Sequential Breakpoints.

Target

The CPU board or system where the program you're debugging runs and where the Periscope pod is installed at the CPU. A debugger "monitor" may also run in the target.

See also Host System.

Trigger Event

The event that caused a hardware breakpoint. For example, if you set a breakpoint on a write to port 20h and started execution of your program, a breakpoint would be generated at the end of the next hardware interrupt. When examining the trace buffer, you'd see the out to port 20h as the last item in the trace buffer, with a sequence number of zero, identifying it as a breakpoint, or trigger, event.

A P P E N D I X C

Hardware Reference

- **Jumpers on the pods**
- **Jumpers on the Model IV (Rev 2) board**
- **386SX Adapter**

This appendix documents the layouts of and jumpers on the 386 and 486 pods, the Model IV board, and the 386SX adapter.

C.1 JUMPERS ON THE PODS

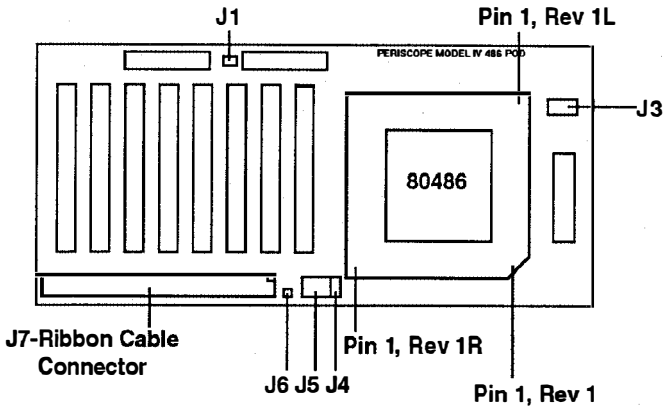


Figure C-1. Layout of the 486 pods

All 486 pods (Rev 1, Rev 1L, and Rev 1R) are identical except for the location of Pin 1.

The jumper definitions below apply to both the 386 and the 486 pods. See Figure C-1 and Figure C-2.

J1 and J3— These pins are always shunted.

J4—EXTERNAL PROBE INPUT. When you hold the pod so that the ribbon cable connector (J7) is on the bottom, pin 1 is the bottom pin. Pin 2 is ground and pin 1 is an external probe input, which must be driven by a TTL-compatible signal. When you set J5 to select the external probe and the external signal is high, the word Probe appears in the hardware trace buffer display.

J5—PROBE SELECT. When you hold the pod so that the ribbon cable connector (J7) is on the bottom, pin 1 is in the bottom right corner. The six J5 pins select the source of the probe bit in the hardware trace buffer. The three valid positions are:

(1) When you connect pins 1 and 2, the CPU Hold Acknowledge signal is the probe bit.

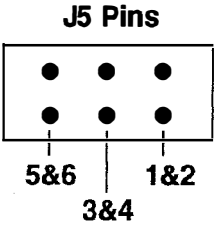


Figure C-2. J5 Pin Assignment on 386 and 486 pods

(2) When you connect pins 3 and 4, the CPU IRQ line is the probe bit (default).
(3) When you connect pins 5 and 6, the external probe input is the probe bit.

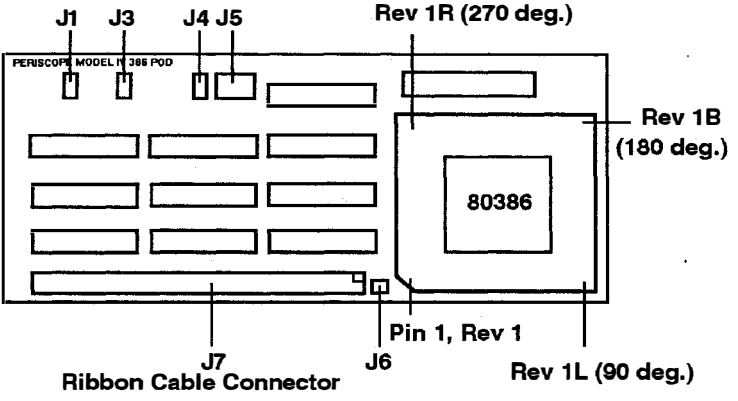


Figure C-3. Layout of the 386 pod

Rotator sockets rotate Pin 1 to the three corners not covered by the 386 Rev 1 pod. Call Periscope Sales for details.

J6—NMI CLIP . Connect this pin to the motherboard to generate an NMI via the Model IV board.

C.2 JUMPERS ON THE MODEL IV (REV 2) BOARD

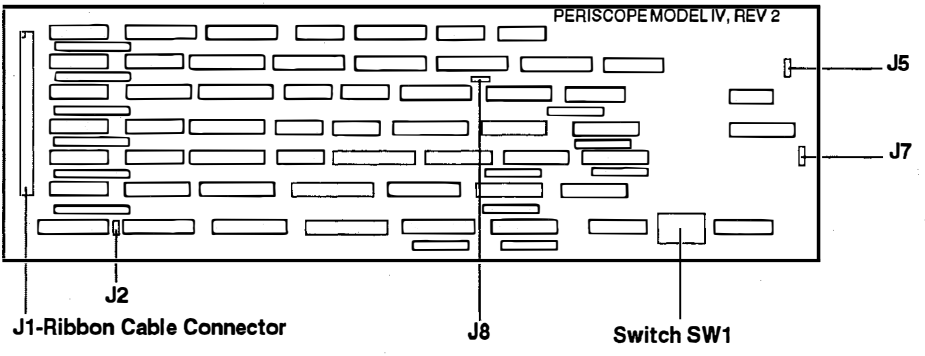


Figure C-4. Layout of the Model IV, Rev 2 board

J2—TERMINATION LINE. These pins are never shunted.

J5—ENABLE LOCAL NMI. These pins are always shunted.

J7—EXTERNAL SIGNAL SELECT. Use this jumper to control the signal source for J6 (pod). When the jumper shunts pins 2 and 3 (the two pins nearest to the gold fingers), the NMI signal is selected (default).

J8 (near R15)—TERMINATION LINE. These pins are always shunted.

SW1—DIP Switch 1. See Setting the DIP Switch in Chapter 3.

C.3 386SX ADAPTER

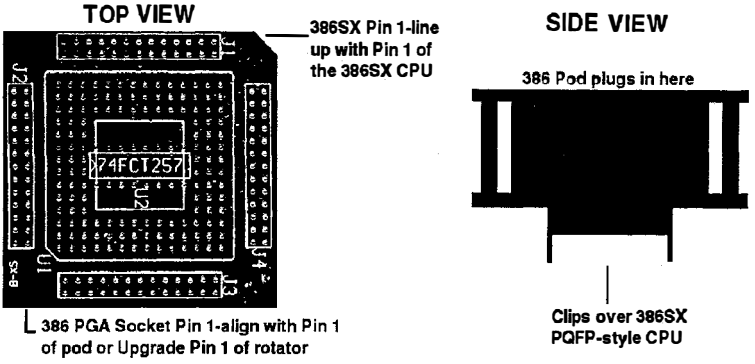


Figure C-5. 386SX Adapter, Top and Side Views

The 386SX adapter shown above is required when the target CPU is a 386SX in a Plastic Quad Flat Pack package. The adapter clips over the CPU, converting it to a Pin Grid Array package that the 386 pod plugs into. The 386SX adapter's dimensions are 2.25" inches square by one inch high (not including the part that clips over the CPU).

WARNING! The 386SX Adapter is an expensive, special-purpose device. Improper installation can damage it beyond repair. **PLEASE** read the installation instructions very carefully before installation, and call Periscope Technical Support for help if you need it!

Index

/4 command, 82
386 Pod Layout, 111
486 Pod Layout, 110
80386, 7, 24, 58, 59, 62, 88, 96
80386EX, 96
80386EX adapter, 96
80386SX, 24, 57, 59, 96
80386SX adapter, 96
80386SX, installation on, 24
80486, 7, 24, 58, 59, 62, 88, 96
80486DX2, 4, 24, 97
80486DX4, 24, 97
80486SX, 97
80486SX2, 97

—A—

Active remote mode, 12, 13, 97
Adapter, 19
Address, 62, 69

—B—

BB command, 47
BBS, 5
BF command, 47
BIOS, 80, 88
Bit breakpoint, 9
Bit mask, 9
Boards, 3, 20, 29, 102
BR command, 47
Break-out Switch, 3, 8, 11, 31, 97
Breakpoint, 98
  Code, 46, 47
  Debug register, 46, 47
  Event, 37
  Hardware, 46, 47
  Monitor, 46
  Overrun, 48, 62, 87, 98
BS-16 Memory, 98
BU command, 47
Buffer capture, 51
Burst mode, 62, 74, 98
Bus compatibility, 87
BW command, 47

—C—

Cache, 82, 98
Capabilities, 8
Capturing specific code execution, 85
Chip puller, 19
Code prefetch, 62
CodeView, 2
Commands
  /4 (toggle internal 486 cache), 45, 82
  GH (Go using Hardware), 46
  GM (Go using Monitor), 47
  HA (Hardware breakpoints All), 49
  HB (Hardware Bit breakpoint), 50
  HC (Hardware Controls), 51
  HD (Hardware Data breakpoint), 56
  HM (Hardware Memory breakpoint), 61

HP (Hardware Port breakpoint), 65
HW (Hardware Write), 80
Hx (display Hardware trace buffer), 66
Compatibility, 5
Conflicts, 20
Controls, 51
CPU, 24, 57, 99
CPU cycle count, 10, 11
CPU event, 10, 57, 99
Crowbar tools, 27
Cycle count, 10, 51, 52, 71, 76, 99

—D—

Data breakpoints, 9, 87
Data field, 69
Debug registers, 46, 100
Diagnostics, 32
DIP switch, 20, 22, 33
Disabling buffer capture, 51
DMA (Direct memory access), 9, 87, 100
DOS usage, 80

—E—

EISA, 100
EM Memory, 100
Error messages, 91
Exception interrupts, 10, 84
Exclude state(s), 51, 54
Extended memory, 76

—F—

Fixup addresses, 77
Flushed instructions, 76
Flushing the trace buffer, 73
Full trace buffer, 10

—G—

GH command, 46
GM command, 45, 47
Guarantees, 5

—H—

HA command, 49
Halt, 100
Hardware breakpoint, 8, 46, 100
Data bit mask, 8
Data values, 8
Examples, 88
I/O port access, 8
Memory access, 8
Pass counter, 8
Selective capture, 9
Sequential triggers, 9
Hardware Commands. *See* Commands
Hardware interrupts, 86
Hardware requirements, 4
Hardware trace buffer, 10, 48, 66, 101
Hardware-assisted debugger, 7
HB command, 50
HC command, 51
HD command, 56
HM command, 61
Host debugger, 101
Host system, 101
HP command, 65
HR command, 66
HS command, 66
HT command, 66
HU command, 66
HW command, 80

—I—

I/O ports, 20
IBM PS/2, 15, 52
Installation overview, 18
Installation tools, 23
Interrupt acknowledge, 101
Interrupt vector reads, 88
Interrupt vector writes, 88
ISA, 102

—J—

Jumpers, 386/486 pods, 110
Jumpers, Rev 2 board, 112

—K—

Key usage, 73

—M—

MCA, 102
Memory breakpoints, 9, 61
  Code prefetch, 61
  CPU halt, 61
  Interrupt Acknowledge, 61
  Read, 61
  Write, 61
Memory Manager, 102
Menus, 44
Micro Channel bus, 4, 15
Missed breakpoint, 48
Mixed mode, 76
Model IV package, 3
Monitor breakpoints, 46, 47

—N—

Network card, 20
NMI, 13, 87, 102
NMI clip, 14, 102, 112
NOTES.TXT, 59
Null-modem Cable, 103

—O—

Operation, 69
Operation field, 70
OS/2 (IBM), 12
Overrun error, 89

—P—

Pass count, 51, 53
Pass counter, 8, 9
Passive remote mode, 12, 103
PC-104 Bus, 103
Periscope ID field, 72
Periscope Model IV, 7
Periscope software installation, 32
Periscope/32, 2
Periscope/EM, 2, 8
PGA (Pin Grid Array), 24, 103
Pin 1, 24, 25, 103
Pipelining, 63, 104
Plus board, 8, 11, 31, 104
Pods, 3, 25, 104
PopUp Periscope, 2, 3
Port breakpoints, 9
Port read, 65
Port write, 65
POST, 12, 85
Post-mortem debugging, 84
Power requirements, 5
Power-on startup tests, 12, 85
PQFP (Plastic Quad Flat Pack), 104
Prefetch queue, 62, 63, 77, 88, 104
Probe, 70, 104
Probe bit, 10, 111
Probe triggering, 51, 52
Protect mode, 14
Protected Memory, 105
Prototype Card, 20
PS4TEST, 32, 80, 91
PSBUF.DAT, 33, 80
PSTEST, 34

—R—

Raw mode, 67
Readme.txt file, 3
Real-time applications, 11
Real-time debugging, 105
Rebooting, 87
Register contents, 74
Remote debugging, 4, 105
Rev 2 Board Layout, 112
Ribbon Cable, 13, 19, 25, 29, 105
RUN, 46, 49

—S—

Save trace buffer, 80
Search trace buffer, 77
Segment decoding, 77
Selective capture, 9, 51
Sequence number field, 72
Sequential triggers, 9, 61, 62, 65, 105
Set cursor calls, 88
SETUP, 32, 36
Single-entry mode, 66, 67
Single-System Mode, 106
SKIP21, 47
Software breakpoints, 8, 106
Software requirements, 4
Software Trace Buffer, 106
Source only mode, 76
State machine, 9, 106
System bus, 57
System requirements, 4

—T—

Target system, 106
Technical support, 5
Trace buffer
  Analysis, 73, 76
  Commands, 75
  Discontinuity, 48
  Display formats, 10
  Display layout, 69
  Menu, 75
Trace mode, 67
Trace overflow stop, 51
Tracking program flow, 84
Trigger event, 107
Trigger location, 51
Turbo Debugger, 2
Tutorial, 36
Typography, 2

—U—

Unassembly mode, 68
Underflowing stack, 88
Upgrades, 5

—W—

Warranties, 5
Windows (Microsoft), 12
Write-protected memory, 8, 11
Writing to low memory, 48

Artisan Technology Group is an independent supplier of quality pre-owned equipment

Gold-standard solutions

Extend the life of your critical industrial, commercial, and military systems with our superior service and support.

We buy equipment

Planning to upgrade your current equipment? Have surplus equipment taking up shelf space? We'll give it a new home.

Learn more!

Visit us at **artisantg.com** for more info on price quotes, drivers, technical specifications, manuals, and documentation.

Artisan Scientific Corporation dba Artisan Technology Group is not an affiliate, representative, or authorized distributor for any manufacturer listed herein.

We're here to make your life easier. How can we help you today?

(217) 352-9330 | sales@artisantg.com | artisantg.com

ARTISAN® TECHNOLOGY GROUP