MicroPower/Pascal™ Debugger User's Guide

AA–M393B–TC

digital

software

MicroPower/Pascal™ Debugger User's Guide

AA–M393B–TC

July 1983

This manual describes how to use the MicroPower/Pascal symbolic debugger, PASDBG. The manual describes the hardware and software required for a PASDBG application, explains the commands used in PASDBG, and presents examples.

This manual supersedes the *MicroPower/Pascal Debugger User's Guide*, AA–M393A–TC.

This manual contains Update Notice 1, AD–M393B–T1.

Operating System: RT–11 Version 5.0
RSX–11M Version 4.1
RSX–11M–PLUS Version 2.1
VAX/VMS Version 3.4

Software: MicroPower/Pascal Version 1.5

digital equipment corporation · maynard, massachusetts

First Printing, February 1982
Updated, June 1982
Updated, October 1982
Updated, February 1983
Revised, July 1983
Updated, February 1984

M27300

UPDATE NOTICE 1

MicroPower/Pascal™ Debugger User's Guide

AD–M393B–T1

February 1984

NEW AND CHANGED INFORMATION

This update contains changes and additions to the *MicroPower/Pascal Debugger User's Guide*, AA–M393B–TC.

digital equipment corporation · maynard, massachusetts

INSTRUCTIONS

The enclosed pages are replacements for or additions to current pages of the *MicroPower/Pascal Debugger User's Guide*. On replacement pages, changes and additions are indicated by vertical bars (▌); deletions are indicated by bullets (●).

Keep this notice in your manual to maintain an up-to-date record of changes.

Old page	**New page**
Title/Copyright	Title/Copyright
iii/iv to vii/viii	iii/iv to vii/viii
1–1/1–2 to 1–19/blank	1–1/1–2 to 1–21/blank
2–1/2–2	2–1/2–2
2–9/2–10	2–9/2–10
	2–10.1/blank
2–17/2–18	2–17/2–18
2–19/blank	2–19/blank
3–1/3–2	3–1/3–2
3–3/3–4	3–3/3–4
3–9/3–10 to 3–15/3–16	3–9/3–10 to 3–15/3–16
	3–16.1/blank
3–17/3–18	3–17/3–18
3–19/3–20	3–19/3–20
	3–20.1/blank
3–27/3–28 to 3–33/3–34	3–27/3–28 to 3–33/3–34
3–41/3–42	3–41/3–42
3–43/3–44	3–43/3–44
3–49/3–50	3–49/3–50
3–69/3–70	3–69/3–70
	3–70.1/blank
A–1/A–2	A–1/A–2
Index–1/Index–2	Index–1/Index–2
Index–3/Index–4	Index–3/Index–4
Reader's Comments/Mailer	Reader's Comments/Mailer

CONTENTS

FIGURES

TABLES

PREFACE

This manual describes the features and use of the MicroPower/Pascal symbolic debugger, PASDBG. The manual is based on the assumption that you are familiar with either Pascal or MACRO-11. In addition, it is assumed that you have read the MicroPower/Pascal installation guide for your host system, Chapter 3 of the MicroPower/Pascal system user's guide for your host system, and Chapter 2 of the *MicroPower/Pascal Runtime Services Manual*. The installation guide describes host and target hardware configuration requirements for debugging; Chapter 3 of the system user's guide describes the application build procedure; and Chapter 2 of the *MicroPower/Pascal Runtime Services Manual* describes data structures that are referenced in this manual.

Chapter Summary

Chapter 1 provides a short description of debugger features and discusses the hardware and software requirements of the debugger. The chapter also outlines the build procedure for debugging, describes the format for debugger commands, and explains how to begin and end a debugging session. The chapter ends with a sample debugging session.

Chapter 2 describes the debugger commands and explains how to use the debugger to load your application program, reference symbol names and addresses, control program execution, examine and modify data, and examine real-time process control structures. The chapter also provides hints for dealing with problems and special situations that can occur while you are debugging.

Chapter 3 provides information about debugger command format, qualifiers, and parameters for quick reference. Sections are ordered alphabetically by command name.

The Appendix describes the protocols that govern communication between the debugger in the host system and the Debugger Service Module (DSM) in the target system. These protocols are transparent to the user.

Associated Documents

The following software documentation is required for complete reference purposes. Refer to the documentation list for your host operating system.

- **RT-11 Host:**

  1. MicroPower/Pascal-RT documentation set. A complete list of documents is contained in the *MicroPower/Pascal-RT Documentation Directory* (Order No. AA-W966B-TC).

2. RT-11 V5 host operating system documentation set. A subset of the RT-11 V5 documentation set is contained in the MicroPower/Pascal-RT documentation set. No additional RT-11 documentation is required for MicroPower/Pascal-RT application software development.

- **RSX-11M/M-PLUS Host:**

1. MicroPower/Pascal-RSX documentation set. A complete list of documents is contained in the MicroPower/Pascal-RSX Installation Guide (Order No. AA-AK1ØA-TC).

2. RSX-11M/M-PLUS host operating system documentation set. Refer to the documentation set supplied with your host operating system.

- **VAX/VMS Host:**

1. MicroPower/Pascal-VMS documentation set. A complete list of documents is contained in the MicroPower/Pascal-VMS Installation Guide (Order No. AA-AI16A-TE).

2. VAX/VMS host operating system documentation set. Refer to the documentation set supplied with your host operating system.

Document Conventions

This document uses the following conventions:

Convention	**Meaning**
[ ]	Square brackets indicate that the enclosed item is optional.
{ }	Braces enclose lists from which one element is to be chosen.
...	A horizontal ellipsis indicates that the preceding item(s) can be repeated.
. . .	A vertical ellipsis indicates that not all the statements in an example or figure are shown.
PASDBG>**load test**	In examples, user input is shown in **boldface** type.
CTRL/x	The symbol CTRL/x means that you should type a key (represented by x) while holding down the key labeled CTRL.
<RET>	The symbol <RET> means that you should press the RETURN key.

Numbers are in decimal radix unless otherwise specified.

Symbol conventions for the operator symbols used in debugger commands are listed at the beginning of Chapter 3.

CHAPTER 1

GETTING STARTED

This chapter introduces the MicroPower/Pascal interactive debugger, PASDBG, and provides the information you need to run it. Section 1.1 lists the debugger features. Section 1.2 describes the hardware and software required to run PASDBG. Section 1.3 outlines the build procedure for debugging. Section 1.4 explains how to invoke and issue commands to the debugger. Section 1.5 shows a sample debugging session.

1.1 DEBUGGER FEATURES

The PASDBG symbolic debugger provides the following features:

- A communication link between the host computer and the target system, allowing you to down-line load and control an application program from a terminal on the host computer
- Commands and qualifiers that allow you to control and examine an executing program
- Access to the symbol table generated by the Pascal compiler, providing symbolic (Pascal language) referencing and variable access
- Access to process control variables and structures
- A mechanism for user control after an execution error

1.1.1 Host/Target Communication

PASDBG resides as a running program on the development, or host, system. You issue commands to PASDBG from a terminal on the host system. Your application program resides on the application, or target, system and contains a module called the debugger service module (DSM). You add the DSM to your application program by using the DEBUG = YES option in your configuration file. A serial line connects the host and target systems, providing a communication path between PASDBG and the DSM. This serial line is used to down-line load the application for debugging and to control the application. (Refer to Section 1.2 and the LOAD command in Chapter 3 for details on down-line loading with PASDBG.) No terminal is required on the target system.

1.1.2 Commands

The debugger commands allow you to control execution of your MicroPower/Pascal program and to examine and modify your program and data. Section 1.4 lists the debugger commands, and Chapters 2 and 3 describe the commands in detail.

1.1.3 Symbolic Debugging

The Pascal compiler produces a table of symbols that PASDBG can reference during a debugging session. You generate these symbols by using the /D switch for those programs or modules that you wish to debug. The symbol table provides information on all variables, structured variables, pointer variables, user-defined types, labels, procedures, functions, main programs, and modules. If the Pascal program uses descriptor variables, the symbol table also contains information about semaphores and ring buffers.

1.1.4 Process Control Variables and Structures

PASDBG permits access to process control variables and structures. For example, you can examine semaphore states, process state queues, and blocked and active processes. Chapter 2 discusses process control variables in more detail.

1.1.5 Execution Error Restart

PASDBG allows you to examine or restart a program after the occurrence of a runtime error. If necessary, PASDBG can down-line load a new copy of the application program. PASDBG can also restart execution from the point where the program failed. If the target system crashes, PASDBG allows you to run the target Micro-ODT from the host terminal. Refer to Section 2.3.6 and the SET ODT command in Chapter 3 for more details on ODT mode.

1.2 THE DEBUGGER ENVIRONMENT

This section describes the hardware and software you need to run PASDBG. This section also tells you how to correct host/target communication problems that you may encounter when you invoke the debugger or down-line load your application program.

NOTE

The hardware descriptions in this section are categorical and not host-specific. For detailed, host-specific descriptions of the host and target hardware requirements for debugging, see the MicroPower/Pascal installation guide for your host system.

1.2.1 Required Hardware

The system configuration for debugging must contain at least the following hardware:

Host System

- PDP-11 or VAX-11 processor with memory management and line clock
- Two asynchronous serial-line ports: one for the terminal and the other for the target system
- A file-structured storage device
- 64K words of memory for RT-11, 124K words for RSX-11M, 256K words for RSX-11M-PLUS, or, for VAX/VMS, sufficient memory as described in the MicroPower/Pascal-VMS V1.5 Software Product Description

Target System

- PDP-11 processor
- Sufficient memory to contain the application program, 4K bytes of RAM minimum
- One serial-line port for the host system
- Other peripheral devices as required by the application program

Figure 1-1 illustrates the configuration required for down-line loading and debugging a target system from an RT-11 host system. For RSX-11M/M-PLUS and VAX/VMS host systems, the configuration for debugging each target system connected to the host is conceptually similar to that shown below. The hardware differences are described in the MicroPower/Pascal installation guides for the respective host systems.

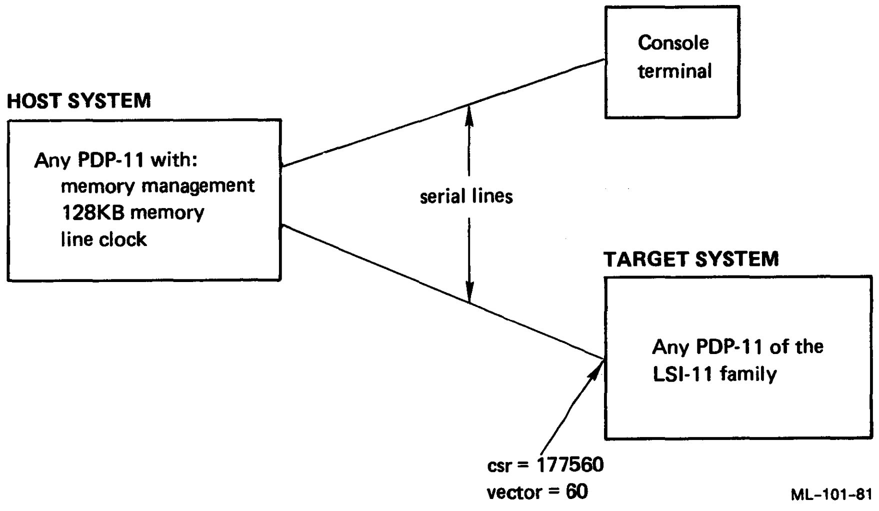

Figure 1-1 PASDBG Hardware Configuration

PASDBG uses a serial line to communicate between the host and target systems. PASDBG down-line loads your application program -- refer to the LOAD command in Chapter 3 -- and controls the application through this serial line. You must configure your host and target hardware in the following way so that PASDBG can use the serial-line link:

1. Connect the host/target serial line to the target system's console line.
2. Set the control status register of the target system's serial-line port to 177560 and its vector to 60.
3. Set the target system's console line to halt the processor on a BREAK command.

NOTE

1. Do not attempt to use PASDBG to debug programs that reside in read-only memory (ROM).
2. Do not use the console serial line for the application program.

1.2.2 Required Software

You need the following programs, files, and libraries to run PASDBG.

Host System

- RT-11, RSX-11M/M-PLUS, or VMS operating system
- The .MIM file created during the application build procedure (application program master copy)
- The .DBG file created during the application build procedure (required for symbolic debugging)
- PASDBG.SAV (RT-11), MPPPDB.TSK (RSX-11M/M-PLUS), or PASDBG.EXE (VMS)
- PASDBG.HLP
- TDBOTU.BOT or TDBOTM.BOT
- TDX.SYS (RT-11 host only; must reside on SY:)
- MPPBRK.TSK (RSX-11M/M-PLUS host only; resides by installation default on MP:[1,54])

The .DBG file created during the application build procedure contains information that PASDBG uses for access to symbols and kernel information. You specify use of the .DBG file with the LOAD command (refer to Chapter 3). The .DBG file is not required for running the debugger, but it is needed for access to symbols in the user program and to kernel symbols. Symbol- or kernel-specific commands, such as EXAMINE FOO and SHOW RUN QUEUE, do not work without the .DBG file.

TDBOTU.BOT and TDBOTM.BOT are used as bootstraps for the PASDGB LOAD command. When the LOAD command is used, it examines the .MIM file to determine whether the file is mapped. If the .MIM file is mapped, PASDBG will use TDBOTM.BOT as the bootstrap. If the .MIM file is not mapped, PASDBG will use TDBOTU.BOT as the bootstrap.

The PASDBG .SAV/.TSK/.EXE and .HLP files and the appropriate TD bootstrap must reside on a particular installation-defined logical device for each type of host system. The installation default for each host is as follows:

Host	**Logical Device**
RT-11	LB: or SY:
RSX-11M/M-PLUS	MP:[1,54] for MPPPDB.TSK MP1:[1,1] for PASDBG.HLP and TD bootstrap
VAX/VMS	MICROPOWER$LIB:

TDX.SYS is the load module for the standard RT-11 serial-line (TD) handler. The TD handler controls the serial line connecting the host and target systems and is used by PASDBG-RT to communicate between host and target. PASDBG-RT requires that the TD handler be loaded; if it is not, PASDBG-RT loads it automatically.

NOTE

PASDBG-RSX uses the standard RSX-11 terminal driver to handle the serial line through which it communicates with the target system. Similarly, PASDBG-VMS uses the standard VMS terminal driver to handle its host/target serial line. Software setup requirements for the serial line are determined by the host operating system and differ somewhat between PASDBG-RSX and PASDBG-VMS. In both cases, however, you must assign the logical device name TD: to the line connected to your target system before invoking PASDBG, and the line must be set to a speed matching that of the target's console terminal port. See Chapter 10 of the _MicroPower/Pascal-RSX/VMS System User's Guide_ for details on host/target line setup for RSX and VMS hosts.

MPPBRK.TSK is a slave task that PASDBG-RSX uses to perform privileged commands to the device TD. The debugger requires that MPPBRK.TSK be installed with the name ...BRK on the host system. When invoked, the debugger checks for ...BRK and issues an error message if it is not installed. The ...BRK task is used for such operations as setting TD line characteristics and sending BREAK signals down the TD line to halt the target for loading or for SET ODT commands. Only the debugger may invoke ...BRK.

Application System

- The .MIM file created during the application build procedure (the application program)
- The debugger service module (DSM) as part of the application program

The PASDBG debugger has two parts: the debugger service module (DSM) and the PASDBG program. The PASDBG program resides on the host system and runs under the host operating system. The DSM resides on the application system as part of the application program.

To use the debugger, you run the PASDBG program and give it commands. PASDBG interprets each command you give it and issues a command to the DSM over the serial-line link between the processors. The DSM, in turn, answers PASDBG over the same line. PASDBG maintains most of the information required for debugging the program, thus minimizing the space required for the DSM on the application system.

1.2.3 **Host/Target Communication Problems**

This section tells you how to correct host/target communication problems you may encounter when you invoke the debugger or down-line load your application program.

1.2.3.1 **No Target Communication** - Under certain conditions, PASDBG may not be able to establish communication between the host system and the target system. (Communication is defined as the target system's responding to a simulated Break sent by the host system by halting and executing console ODT microcode.) If communication cannot be established, PASDBG displays the following error message:

```
?PASDBG-E-NOCOMM, No target communication (check power and
cables)
```

If this error message is displayed, make sure that:

- The target system is powered up
- The serial line between the host and target systems is connected
- The serial line baud rates are equal in both the host and target systems
- The communications line is attached to the console serial I/O port of the target system
- The target system hardware is configured to respond to Break by entering ODT
- (RT-11 host only:) The TD handler is set for the correct CSR and interrupt vector values -- see Section 1.2.3.3
- (RSX-11M/M-PLUS or VAX/VMS host only:) The system-dependent parameters of the host/target link are set up as described in Chapter 10 of the MicroPower/Pascal-RSX/VMS System User's Guide

1.2.3.2 **No Communication with FALCON or FALCON-PLUS Target** - If your target system is an SBC-11/21 and is running at processor priority 7, you must turn the target off and on again to establish communication. PASDBG cannot halt an SBC-11/21 if the processor priority is 7.

If your target system is an SBC-11/21 and if the value specified for the configuration file KXT11 macro baud rate parameter is not the same as the line rate, the target system will load correctly but will fail with the "No target communication" message once the application program initializes. You can correct this problem by changing the KXT11 macro slu1 parameter to the correct (line) baud rate.

1.2.3.3 **Nonstandard Vector and CSR on RT-11 Host** - If your target system is attached to an RT-11 host system by a serial-line interface whose interrupt vector and CSR are not configured to 300 and 176500, respectively, use the RT-11 commands SET TD VECTOR and SET TD CSR on the host system to set the correct vector and CSR values. You must first UNLOAD and REMOVE the TD handler before typing the SET commands; then INSTALL and LOAD the TD handler before running PASDBG.

1.2.3.4 **Target System Halts on Load Command** - If the PASDBG bootstrap fails to execute properly after a LOAD command has been issued, the target system halts, and the following error message sequence is displayed:

```
     ?PASDBG-E-TARHALT, Target halted - PASDBG in uODT mode
     ?PASDBG-I-NODSM, No DSM, target not yet loaded
     @
```

The probable cause of this error is that the system configuration file used when building the kernel did not accurately describe the target memory configuration. For example, the MEMORY macro(s) specified more RAM than is available on the target or did not correctly specify the base addresses of noncontiguous memory segments. You must modify the configuration file and rebuild the application image or reconfigure the target hardware to match the description in the configuration file.

1.2.3.5 **TD Logical Device Access on RSX or VMS Host** - If you invoke PASDBG on an RSX11-M/M-PLUS or VAX/VMS host system and PASDBG cannot access the logical device TD, the following error message is displayed:

```
     ?PASDBG-E-NOTD, Unable to access communication line, TD
```

The probable cause of this error is either that you did not assign the logical device name TD: to the line connected to your target system or that the line assigned as TD: is allocated to or in use by another user. On a VAX/VMS host system, the error also occurs if you do not have read/write access to the line assigned as TD:.

1.3 BUILD PROCEDURE FOR DEBUGGING

This section outlines the build procedure you must follow in order to debug an application program with PASDBG. It is assumed that you have read Chapter 3 of the MicroPower/Pascal System User's Guide for your host system. That chapter gives more complete information on the application build cycle and the full range of build options.

In general, building an application for debugging requires the following special steps:

1. Build a kernel with the debugger service module (DSM) included. To do that, specify debug=YES in your configuration file:

   ```
   SYSTEM optimize=NO, debug=YES

                   or

   SYSTEM optimize=YES, debug=YES
   ```

2. If you are using the DIGITAL-supplied command procedure MPBUILD (RSX or VMS) or MPBLD (RT-11) to build your application, build debugger support into your application. You do this by answering "yes" to the debug support question in the MPBUILD/MPBLD dialog:

   ```
   Debug support required [yes] ? yes<RET>
   ```

   If you use MPBUILD/MPBLD, no other special steps are necessary, except step 4 if your application requires the XL handler. Ignore steps 3 and 5 through 9 in this list. The MPBUILD/MPBLD command procedure is described in Appendix B of the MicroPower/Pascal system user's guide for your host system.

3. Build the kernel symbols into the application. To do that, use the debug switch on the kernel module while running MERGE and RELOC and specify a DBG file for the kernel in MIB.

   RT-11 Host Example

   ```
   .R MACRO<RET>
   *KRNM=KRNM,LB:COMM.SML/M<RET>

   .RUN LB:MERGE<RET>
   *KRNM.MOB/D=KRNM,LB:PAXM<RET>
   *<CTRL/C>

   .RUN LB:RELOC<RET>
   *KRNM,KRNM,KRNM/D=KRNM.MOB<RET>
   *<CTRL/C>

   .RUN LB:MIB<RET>
   *KRNM,KRNM.MIB,KRNM=KRNM,,KRNM/K/S<RET>
   *<CTRL/C>
   ```

RSX-11M/M-PLUS Host Example

```
>MAC<RET>
MAC>KRNM=MP:[2,10]COMM.MLB/ML,dev:[uic]KRNM<RET>

>MRG<RET>
MRG>KRNM.MOB=KRNM,MP:[2,10]PAXM/LB/DE<RET>

>REL<RET>
REL>KRNM,KRNM,KRNM/DE=KRNM.MOB<RET>

>MIB<RET>
MIB>KRNM,KRNM.MIB,KRNM=KRNM,,KRNM/KI/SM<RET>
```

VAX/VMS Host Example

```
$MCR MAC<RET>
MAC>KRNM=MICROPOWER$LIB:COMM.MLB/ML,dev:[dir]KRNM<RET>

$MPMERGE<RET>
MRG>KRNM.MOB=KRNM,MICROPOWER$LIB:PAXM/LB/DE<RET>

$MPRELOC<RET>
REL>KRNM,KRNM,KRNM/DE=KRNM.MOB<RET>

$MPMIB<RET>
MIB>KRNM,KRNM.MIB,KRNM=KRNM,,KRNM/KI/SM<RET>
```

4. If your application requires the XL handler, use the prefix file XLPFXD.MAC instead of XLPFX.MAC to add the handler to your application. This ensures that the application will not use interrupt vector 60 and CSR 177560, which are reserved for the host/target serial line. (For SBC-11/21 applications, use the prefix file XLPFXF.MAC; for KXT11-C applications, use XLPFXK.MAC.)

   RT-11 Host Example

   ```
   .R MACRO<RET>
   *XLPFXD=XLPFXD.MAC,LB:COMM.SML/M<RET>
   ```

   RSX-11M/M-PLUS Host Example

   ```
   >MAC<RET>
   MAC>XLPFXD=MP:[2,10]COMM.MLB/ML,dev:[uic]XLPFXD.MAC<RET>
   ```

   VAX/VMS Host Example

   ```
   $MCR MAC<RET>
   MAC>XLPFXD=MICROPOWER$LIB:COMM/ML,dev:[dir]XLPFXD<RET>
   ```

   If you are using the MPBUILD/MPBLD command procedure to build an application that requires the XL handler, specify the XLPFXD prefix file (XLPFXF for SBC-11/21, XLPFXK for KXT11-C) in response to the handler prefix question in the MPBUILD/MPBLD dialog:

   ```
   Handler prefix file spec ? XLPFXD.MAC<RET>

   Is this a Pascal-Implemented handler [no] ? <RET>
   ```

5. Generate the symbols for the Pascal modules you wish to debug. To do so, use the debug switch switch in the compiler command line.

   RT-11 Host Example

   ```
   .R PASCAL<RET>
   *ACE=ACE/D<RET>
   ```

   RSX-11M/M-PLUS Host Example

   ```
   >MPP<RET>
   MPP>ACE=ACE/DE<RET>
   ```

   VAX/VMS Host Example

   ```
   $MPPASCAL<RET>
   MPP>ACE=ACE/DE<RET>
   ```

   For a MACRO-11 static process or routine that you wish to debug, you generate the ISD records for the module's global symbols not at assembly time but at merge time, via the MERGE debug option (see step 6). Do not use the assembler debug option (E:DBG), as it generates debug information in a format incompatible with PASDBG.

   RT-11 Host Example

   ```
   .R MACRO<RET>
   *ACE=ACE,LB:COMM.SML/M<RET>
   ```

   RSX-11M/M-PLUS Host Example

   ```
   >MAC<RET>
   MAC>ACE=MP:[2,10]COMM.MLB/ML,dev:[uic]ACE<RET>
   ```

   VAX/VMS Host Example

   ```
   $MCR MAC<RET>
   MAC>ACE=MICROPOWER$LIB:COMM.MLB/ML,dev:[dir]ACE<RET>
   ```

6. Include the debug symbols for the modules you wish to debug in the MOB file. To do that, use the debug switch in the MERGE command line. If you are merging a MACRO-11 static process, you can omit the Pascal object-time library (LIBxxx) from the command line.

   RT-11 Host Example

   ```
   .RUN LB:MERGE<RET>
   *ACE=ACE/D,KRNM.STB,LB:LIBNHD<RET>
   *<CTRL/C>
   ```

   RSX-11M/M-PLUS Host Example

   ```
   >MRG<RET>
   MRG>ACE=ACE/DE,KRNM.STB,MP:[2,10]LIBNHD/LB<RET>
   ```

   VAX/VMS Host Example

   ```
   $MPMERGE<RET>
   MRG>ACE=ACE/DE,KRNM.STB,MICROPOWER$LIB:LIBNHD/LB<RET>
   ```

7. Relocate the debug symbols along with the code. To do that, use the debug switch in the RELOC command line.

   RT-11 Host Example

   ```
   .RUN LB:RELOC<RET>
   *ACE,ACE,ACE=ACE/D<RET>
   *<CTRL/C>
   ```

   RSX-11M/M-PLUS Host Example

   ```
   >REL<RET>
   REL>ACE,ACE,ACE=ACE/DE<RET>
   ```

   VAX/VMS Host Example

   ```
   $MPRELOC<RET>
   REL>ACE,ACE,ACE=ACE/DE<RET>
   ```

   If you are relocating a MACRO-11 static process and if you did not name your program at source level via a .TITLE statement, you must explicitly name the program. (Use the name switch.) The name you specify must match the static process name you specified at source level in the DFSPC$ (Define Static Process) macro call for the process.

   RT-11 Host Example

   ```
   .RUN LB:RELOC<RET>
   *ACE,ACE,ACE=ACE/D/N<RET>
   Program Name?  ACE<RET>
   *<CTRL/C>
   ```

   RSX-11M/M-PLUS Host Example

   ```
   >REL<RET>
   REL>ACE,ACE,ACE=ACE/DE/NM:ACE<RET>
   ```

   VAX/VMS Host Example

   ```
   $MPRELOC<RET>
   REL>ACE,ACE,ACE=ACE/DE/NM:ACE<RET>
   ```

   NOTE

   Use the name switch only if you did not name your program at source level. To name your program at source level, you must include a .TITLE statement at the beginning of the first application module that is to be linked by MERGE. The name you specify in your initial .TITLE statement must match the static process name you specified in the DFSPC$ macro call for the process.

8. Copy or rename the kernel DBG file generated in step 3 into a file with the same name as the final MIM file you wish to produce but with the DBG extension.

   For the example build in this section, the file KRNM.DBG would be copied or renamed to ACE.DBG.

9. Produce a DBG file with your MIM file. To do that, specify a DBG file in the MIB command line.

   RT-11 Host Example

   ```
   .RUN LB:MIB<RET>
   *ACE,ACE.MIB,ACE.DBG=ACE,KRNM,ACE/S<RET>
   *<CTRL/C>
   ```

   RSX-11M/M-PLUS Host Example

   ```
   >MIB<RET>
   MIB>ACE,ACE.MIB,ACE.DBG=ACE,KRNM,ACE/SM<RET>
   ```

   VAX/VMS Host Example

   ```
   $MPMIB<RET>
   MIB>ACE,ACE.MIB,ACE.DBG=ACE,KRNM,ACE/SM<RET>
   ```

1.4 BEGINNING A DEBUGGING SESSION

This section explains how to invoke the PASDBG symbolic debugger and how to issue debugger commands. The section ends with a list of debugger commands.

On an RT-11 host system that has hard-disk storage, PASDBG resides by default on the RT-11 logical disk LB:. You invoke PASDBG with the command:

```
.RUN LB:PASDBG<RET>
```

On a floppy-diskette-only RT-11 host system, PASDBG resides on SY:. You invoke PASDBG with either of the following commands:

```
.RUN SY:PASDBG<RET>

.R PASDBG<RET>
```

On an RSX-11M/M-PLUS host system, PASDBG resides by default in logical device/directory MP:[2,10]. You invoke PASDBG with the command:

```
>PDB<RET>
```

On a VAX/VMS host system, PASDBG resides by default in logical device/directory MICROPOWER$LIB:. You invoke PASDBG with the command:

```
$PASDBG<RET>
```

NOTE

On an RSX-11M/M-PLUS or VAX/VMS host system, you must assign the host/target communication line and correctly set its characteristics before invoking PASDBG. See Chapter 10 of the _MicroPower/Pascal-RSX/VMS System User's Guide_ for instructions on doing so. On a VAX/VMS host system, you must also execute the MPSETUP command file, as directed in Chapter 1 of the _MicroPower/Pascal-RSX/VMS System User's Guide_.

PASDBG responds with a debugger version number, a message reporting the state of the target system, and the following prompt:

```
     PASDBG>
```

The prompt signifies that PASDBG is waiting for a user command. The format for debugger commands is as follows:

```
     PASDBG>command [!comment]<RET>
```

In the command line above, "PASDBG>" is the debugger command-mode prompt, "command" is a debugger command, and "comment" is an optional comment that is echoed but ignored by PASDBG.

The first command issued in a debugging session is usually a LOAD command, a LOG command, a HELP command, or an @ command. LOAD down-line loads the application program and/or loads application symbols, LOG opens a log file, HELP requests information about debugger commands, and @ invokes an indirect command file.

To end a debugging session, issue the EXIT command as follows:

```
     PASDBG>exit<RET>
```

Table 1-1 lists the debugger commands with their associated keywords.

Table 1-1
Debugger Commands

Command	Description
@	Invoke indirect command file
CANCEL BREAK /ALL	Cancel breakpoint(s)
CANCEL PROCESS	Cancel process and mapping
CANCEL SCOPE	Cancel lexical scope
CANCEL STEP	Cancel SET STEP parameters
CANCEL TRACE /ALL	Cancel tracepoint(s)
CANCEL WATCH /ALL	Cancel watchpoint(s)
CLOSE	Close log file
CTRL/C	Exit
CTRL/O	Suppress terminal output
CTRL/Y	Exit (VAX/VMS host only)

(Continued on next page)

Table 1-1 (Cont)
Debugger Commands

Command	Description
DEPOSIT /BYTE	Deposit data
EXAMINE /ASCII /BINARY /BYTE /DECIMAL /HEXADECIMAL /INSTRUCTION /OCTAL /RAD5Ø /REAL /WORD	Examine memory location(s)
EXIT	Exit debugger
GO /EXIT	Start/continue execution
HALT	Stop target
HELP	Display help text
INIT /RESTART	Initialize application
LOAD /EXIT /SYMBOL /TARGET	Load application and/or symbols
LOG	Open log file
SET BREAK /AFTER /PROCESS	Set breakpoint
SET ODT	Enter target ODT
SET PHYSICAL	Set physical addressing
SET PROCESS	Set process mapping
SET PROGRAM	Set upper-level scope
SET SCOPE	Set lower-level scope
SET STEP INSTRUCTION INTO OVER STATEMENT	Set step parameters

(Continued on next page)

Table 1-1 (Cont)
Debugger Commands

Command	Description
SET TRACE /AFTER /PROCESS	Set tracepoint
SET WATCH /AFTER	Set watchpoint
SHOW BREAK	List breakpoints
SHOW CALLS	Display procedure call chain
SHOW EXCEPTION	Display last exception
SHOW EXCEPTION GROUPS	Display exception handling
SHOW FREE PACKETS	Display available packets
SHOW FREE STRUCTURES	Display available kernel pool
SHOW INACTIVE QUEUE	List aborted processes
SHOW NAMES	List named kernel structures
SHOW PACKET QUEUE	Display packet queue
SHOW PCB	Display process information
SHOW PROCESS /ALL	Display mapping or list process(es)
SHOW READY/ACTIVE QUEUE	List ready-active processes
SHOW READY/SUSPENDED QUEUE	List ready-suspended processes
SHOW RING BUFFER	Display ring buffer
SHOW RUN QUEUE	Display running process
SHOW SCOPE	Display current lexical scope
SHOW SEMAPHORE	Display binary/counting semaphore
SHOW STEP	List step parameters
SHOW STRUCTURE	Display program/process/routine/variable structure
SHOW TARGET	Display state of application
SHOW TRACE	List tracepoints
SHOW WATCH	List watchpoints

(Continued on next page)

Table 1-1 (Cont)
Debugger Commands

Command	Description
SPAWN	Execute host command without exiting (RSX-11M/M-PLUS or VAX/VMS host only)
STEP /INSTRUCTION /INTO /OVER /STATEMENT	Execute incrementally

1.5 SAMPLE DEBUGGING SESSION

This section presents a sample debugging session. The session does not locate errors in the sample program used but illustrates some of the basic debugger functions. The commands used in this debugging session -- including the invocation of the debugger -- are shown in **boldface** type. See Section 1.4 and Chapter 3 for alphabetical lists of the debugger commands.

The application used for this debugging session was built from the following files:

File	**Description**
PROCS2.PAS	Pascal source file
CFDUNM.MAC	Configuration file for debugging
COMU.SML/.MLB	System macro library file (unmapped)
PAXU.OBJ/.OLB	Kernel module library file (unmapped)
XLPFXD.MAC	XL driver prefix file for debugging
DRVU.OBJ/.OLB	Driver library file (unmapped)
LIBNHD.OBJ/.OLB	OTS module library file (unmapped)

The MicroPower/Pascal compiler generated the following listing of the user-process source code in PROCS2.PAS:

```
        Line    Stmt   Source
        ---------------------
          1             [ SYSTEM (MICROPOWER) ] PROGRAM procs2;
          2             VAR j,k : INTEGER;
          3
          4             PROCESS pl;
          5             VAR i : INTEGER;
          6             BEGIN
          7            1  FOR i := 0 TO 100 DO
          8                  BEGIN
          9            2     WRITELN ('in pl, i=',i:3);
         10                  END
         11            3 END;
         12
         13             LABEL 10;
         14             BEGIN
         15            1     pl;
         16            2     k := 2;
         17            3     j := 100;
         18            4 10: j := j-1;
         19            5     WRITELN('in main block, j=',j:3);
         20            6     IF j>0 THEN GOTO 10;
         21            7     WRITELN('Process test finished');
         22            8 END.
```

The Pascal statement numbers listed in the "Stmt" column above can be used in debugger commands to reference locations within the program. Note that the statement numbers repeat for each process, procedure, or function. While debugging, you distinguish between identical statement numbers by issuing the SET SCOPE command. (See Chapter 3.)

The debugging session begins with the invocation of PASDBG below. For an RSX-11M/M-PLUS or VAX/VMS host, it is assumed that you have correctly set up the host/target communication line and assigned it the logical name TD:, as directed in Chapter 10 of the <u>MicroPower/Pascal-RSX/VMS System User's Guide</u>.

```
.RUN LB:PASDBG<RET>      or      >PDB<RET>      or      $PASDBG<RET>
PASDBG Vxx.xx      <-- Version number displayed here
;PASDBG-I-NODSM, No DSM, target not yet loaded

PASDBG>log dialog.log<RET>    !Log debugging session

PASDBG>load/target procs2<RET>    !Down-line load application program
;PASDBG-I-BOTWARN, Starting primary boot load, please wait...
;PASDBG-I-BOTLD, Primary boot loaded, getting closer...

 Target stopped at physical (00007716), virtual (007716) : JMP @#10676
 Executing KERNEL code
 No process set, KERNEL mapping in effect

PASDBG>show target<RET>    !Display state of application

 Target stopped at physical (00007716), virtual (007716) : JMP @#10676
 Executing KERNEL code
 No process active
 Not using memory management hardware
```

```
PASDBG>show scope<RET>
 Program KERNEL
;PASDBG-I-KSYNAC, Mapping set, KERNEL symbols not available

PASDBG>show process<RET>
 No process set, KERNEL mapping in effect

PASDBG>load/symbol procs2<RET>   !Get Pascal and KERNEL symbols

 Target stopped at physical (00007716), virtual (007716) : JMP @#10676
 Executing KERNEL code
 No process set, KERNEL mapping in effect

PASDBG>show scope<RET>
 Program KERNEL

PASDBG>set program procs2<RET>   !Set lexical scope to program
;PASDBG-I-NOMOD, Module and Scope set to "PROCS2"

PASDBG>show scope<RET>
 Program PROCS2, Module PROCS2, Scope: PROCS2

PASDBG>show process<RET>   !Display currently set process
 Process  6, name = 'PROCS2', PCB at KERNEL (035436)

PASDBG>show step<RET>
 Step parameters: into statement

PASDBG>set step over statement<RET>

PASDBG>set break 1<RET>   !Set break on stmt 1 in PROCS2

PASDBG>set watch k<RET>   !Set watch on PROCS2 variable

PASDBG>set trace label 10<RET>   !Set trace on PROCS2 label

PASDBG>set scope p1<RET>   !Set lexical scope to process P1

PASDBG>set break 1 'top of p1'<RET>

PASDBG>show break<RET>
  #  Physaddr  AFTER  count   process S.N.   TAG
  0  00053012    1      0        (any)       '1 PROCS2'
  2  00052720    1      0        (any)       'top of p1'

PASDBG>show trace<RET>
  #  Physaddr  AFTER  count   process S.N.   TAG
  1  00053104    1      0        (any)       'LABEL 10 PROCS2'

PASDBG>show watch<RET>
  #  Physaddr  AFTER  count   process S.N.   TAG
  0  00053304    1      0        (any)       'K'

PASDBG>go<RET>   !Start program execution
 [Target execution resumed - type <CR> to stop target]
 ** BREAKPOINT #0  '1 PROCS2'

 Target stopped at physical (00053012), virtual (053012) : MOV
 @#53300,-(SP)
 In statement 1 + 0 in
 Program PROCS2, Module PROCS2, Scope: PROCS2
 Process  6, name = 'PROCS2', PCB at KERNEL (035436)
```

```
PASDBG>go<RET>    !Continue program execution
 [Target execution resumed - type <CR> to stop target]
 ** WATCHPOINT #0  'K'
   Old contents: 0
   New contents: 2

 Target stopped at physical (00053012), virtual (053012) : MOV
 @#53300,-(SP)
 In statement 3 + 0 in
 Program PROCS2, Module PROCS2, Scope: PROCS2
 Process  6, name = 'PROCS2', PCB at KERNEL (035436)

PASDBG>cancel watch k<RET>

PASDBG>go<RET>    !Stop target with <RET>
 [Target execution resumed - type <CR> to stop target]
<RET>
 Target stopped at physical (00053076), virtual (053076) : MOV
 #144,@#53302
 Executing non-Pascal code
 No process set, KERNEL mapping in effect

PASDBG>examine k<RET>   !Show that scope was reset by PASDBG
?PASDBG-E-SYMNDF, Symbol not defined in current scope

PASDBG>show scope<RET>   !In non-Pascal code, no scope
?PASDBG-E-NOSCOPE, No scope set

PASDBG>set program procs2<RET>   !Reset scope
PASDBG-I-NOMOD, Module and Scope set to "PROCS2"

PASDBG>examine k<RET>   !Display contents of variable K
053304 : 2

PASDBG>examine j<RET>   !Display loop-control variable
053302 : 100

PASDBG>go<RET>    !Interrupt with <RET> again
 [Target execution resumed - type <CR> to stop target]
 ** TRACEPOINT #1  'LABEL 10 PROCS2'
 ** TRACEPOINT #1  'LABEL 10 PROCS2'
 ** TRACEPOINT #1  'LABEL 10 PROCS2'
 ** TRACEPOINT #1  'LABEL 10 PROCS2'
<RET>
 Target stopped at physical (00010440), virtual (010440) : MOV
 SP,@#32022
 Executing non-Pascal code
 Process  6, name = 'PROCS2', PCB at KERNEL (035436)

PASDBG>show scope<RET>  !Show scope was updated (cancelled)
?PASDBG-E-NOSCOPE, No scope set

PASDBG>s<RET>    !Step to next statement (reset scope)

 Target stopped at physical (00053146), virtual (053146) : TST @#53302
 In statement 7 + 0 in
 Program PROCS2, Module PROCS2, Scope: PROCS2
 Process  6, name = 'PROCS2', PCB at KERNEL (035436)

PASDBG>show scope<RET>
 Program PROCS2, Module PROCS2, Scope: PROCS2

PASDBG>examine j<RET>   !Examine loop-control variable
053302 : 96
```

```
PASDBG>examine 6<RET>   !Display instruction at stmt 6
053110 :  MOV  #53224,-(SP)

PASDBG>examine 5..7<RET>   !Disassemble stmts 5..6 in PROCS2
053104 :  DEC   @#53302
053110 :  MOV   #53224,-(SP)
053114 :  MOV   #21,-(SP)
053120 :  MOV   (SP),-(SP)
053122 :  JSR   PC,45074
053126 :  MOV   @#53302,-(SP)
053132 :  MOV   #3,-(SP)
053136 :  JSR   PC,45252
053142 :  JSR   PC,45042
053146 :  TST   @#53302

PASDBG>ex %r4<RET>   !Display register 4 contents
R4 :  0

PASDBG>show break<RET>   !List currently set breakpoints
  #  Physaddr  AFTER  count   process S.N.  TAG
  0  00053012    1      0      (any)        '1 PROCS2'
  2  00052720    1      0      (any)        'top of pl'

PASDBG>cancel break #0<RET>   !Cancel breakpoint number 0

PASDBG>set break 6<RET>   !Set break at stmt 6 in PROCS2

PASDBG>go<RET>
 [Target execution resumed - type <CR> to stop target]
 ** TRACEPOINT #1  'LABEL 10 PROCS2'
 ** BREAKPOINT #0  '6 PROCS2'

 Target stopped at physical (00053110), virtual (053110) : MOV
 #53224,-(SP)
 In statement 6 + 0 in
 Program PROCS2, Module PROCS2, Scope: PROCS2
 Process  6, name = 'PROCS2', PCB at KERNEL (035436)

PASDBG>cancel break #0<RET>   !Cancel the break just triggered

PASDBG>examine j<RET>   !Recheck loop control variable
053302 : 94

PASDBG>deposit j=1<RET>   !Force loop to terminate

PASDBG>g<RET>   !Should trigger Pl breakpoint
 [Target execution resumed - type <CR> to stop target]
 ** BREAKPOINT #2  'top of pl'

 Target stopped at physical (00052720), virtual (052720) : CLR R4
 In statement 1 + 0 in
 Program PROCS2, Module PROCS2, Scope: Pl
 Process  9, (no name), PCB at KERNEL (035116)

PASDBG>show scope<RET>   !Lexical scope has been reset
 Program PROCS2, Module PROCS2, Scope: Pl

PASDBG>show process<RET>   !Mapping has been reset
 Process  9, (no name), PCB at KERNEL (035116)
```

```
PASDBG>show structure p1<RET>    !Display structure of P1
  PROCESS P1
    NAME : ARRAY [1..6] OF
       : CHAR
    PRIORITY : INTEGER
    STACK_SIZE : INTEGER
    DESC : INTEGER
    I : INTEGER

PASDBG>show break<RET>    !List breakpoints
  #  Physaddr  AFTER  count   process S.N.  TAG
  2  00052720    1      0      (any)        'top of p1'

PASDBG>init<RET>    !Reinitialize STEP, BREAK, WATCH, etc.

 Target stopped at physical (00052720), virtual (052720) : CLR R4
 In statement 1 + 0 in
 Program PROCS2, Module PROCS2, Scope: P1
 Process  9, (no name), PCB at KERNEL (035116)

PASDBG>show step<RET>    !See that default parameters were reset
 Step parameters: into statement

PASDBG>init/rest<RET>   !Reinit application to starting point

 Target stopped at physical (00007716), virtual (007716) : JMP @#10676
 Executing KERNEL code
 No process set, KERNEL mapping in effect

PASDBG>exit<RET>    !Close log file and exit to monitor
```

CHAPTER 2

DEBUGGING TECHNIQUES

This chapter surveys the general-purpose debugging features PASDBG provides (Section 2.1), describes the PASDBG features that allow you to debug real-time programs (Section 2.2), and presents practical hints and suggestions to keep in mind when debugging MicroPower/Pascal programs (Section 2.3).

PASDBG has five types of commands:

1. Commands that load your application and its symbols for symbolic debugging
2. Commands that establish a context -- lexical scope and mapping -- for symbol and address references in the debugger commands you issue
3. Commands that control the execution of your program
4. Commands that allow you to examine and modify your program and data
5. Commands that allow you to examine the states and control structures in your program

The first four types of commands are discussed in Section 2.1. The fifth type is discussed in Section 2.2. Refer to Chapter 3 for information on command format, parameters, and qualifiers and for examples of command use.

2.1 GENERAL DEBUGGING

This section describes PASDBG's general-purpose debugging commands. These commands load your application and its symbols, establish a context for references to symbols and addresses, control program execution, and examine or modify your program.

2.1.1 Loading the Application Program

The LOAD command down-line loads a copy of the application program into the target computer and/or establishes access to the application's symbol table for symbolic debugging. The LOAD command qualifiers /TARGET, /SYMBOL, and /EXIT select the specific type of load operation to be performed.

LOAD/TARGET
Down-line loads a copy of the application program into the target system.

LOAD/SYMBOL
Loads a copy of the application program's symbol table into PASDBG. PASDBG uses this symbol table to access symbols in the user's program -- including statement numbers, label numbers, and names of programs, modules, processes, procedures, functions, and variables -- and to access kernel symbols. Without this table, PASDBG cannot resolve symbol references, and symbol- and kernel-specific commands, such as EXAMINE FOO and SHOW RUN QUEUE, do not work.

LOAD
Performs a LOAD/TARGET, a LOAD/SYMBOL, or both, depending on the file extension you give in the command line -- .MIM, .DBG, or none specified, respectively.

LOAD/EXIT
Down-line loads a copy of the application program into the target system, starts the application, and then exits to the host operating system, leaving the target running.

2.1.2 Controlling Symbol and Address Resolution

The following commands are used to establish a context -- lexical scope and mapping -- for symbol and address references in the debugger commands you issue.

```
SET PROGRAM
SET PHYSICAL
SET SCOPE
SHOW SCOPE
CANCEL SCOPE
SET PROCESS
SHOW PROCESS
CANCEL PROCESS
```

2.1.2.1 **PROGRAM, SCOPE, and SET PHYSICAL** - The PROGRAM and SCOPE commands establish a lexical context that allows you to reference a symbol in your application program without ambiguity. With SET PROGRAM and SET SCOPE, you can tell the debugger exactly which set of symbols to use when interpreting symbol references in future debugger commands. This makes it possible for the debugger to distinguish between multiple occurrences of a symbol name in your application. You must issue at least a SET PROGRAM command if you want to access Pascal or kernel symbols in your application.

SET PROGRAM
Sets lexical scope to a Pascal or a MACRO-11 program in your application. PASDBG will use the specified program's symbols to resolve future symbolic references in debugger commands. You must issue a SET PROGRAM in order to reference Pascal or MACRO-11 symbols with PASDBG; otherwise, you can access only physical locations within your application.

The \MODULE option on a SET PROGRAM lets you set lexical scope to a particular Pascal module within a particular program. (A Pascal module can appear more than once in an application.) The default for \MODULE is the program -- the module that contains the program declaration.

SET PROGRAM KERNEL sets lexical scope to the kernel. This allows you to reference kernel symbols.

SET PROGRAM performs an implicit SET PROCESS to the program's main (static) process or, if KERNEL is specified, to the kernel. In a virtual system, this establishes process, or kernel, mapping. (See the PROCESS commands, below.)

SET PROGRAM resets the scope set with SET SCOPE to the specified \MODULE. If no module is specified, scope is reset to the main block of the program.

If PROGRAM is not set, PASDBG can access and modify physical locations in memory -- up to 64K bytes for unmapped systems and up to 4M bytes for mapped systems. If PROGRAM is set, PASDBG can access variables within the specified program's memory space symbolically or by virtual address.

SET PHYSICAL

Cancels program, scope, and process settings and sets PASDBG to physical addressing mode. Symbolic access is disabled.

NOTE

The program setting can change whenever the application stops. If the application stops in non-Pascal code -- in the kernel, the OTS, or a MACRO-11 program -- PASDBG cancels the program setting. If the application stops in a different program, PASDBG resets the scope to the currently running program, procedure, function, or process. Because the program setting is subject to change, PASDBG displays the current program, process, and scope settings whenever execution is interrupted.

SET SCOPE

Establishes a context for symbolic references at the procedure, function, or process level. SET SCOPE sets lexical scope to a procedure, function, or process within the currently set program\module (see SET PROGRAM). PASDBG will use the symbols of the specified procedure, function, or process -- or, if necessary, will search upward through the lexical path that leads to the procedure, function, or process -- to resolve future symbolic references in debugger commands. Setting lexical scope properly with SET PROGRAM and SET SCOPE allows you to reference the symbols in your application without ambiguity.

Lexical scope must be set at the program or module level before you set scope to a procedure, function, or process. Use SHOW SCOPE to see if program is currently set. When you set program, scope defaults to the main block of the program or module. When you cancel program (with SET PHYSICAL), scope is also canceled.

NOTE

Whenever the debugger stops the target system, scope is reset to the currently running program, procedure, function, or process, if any. If the application stops in non-Pascal code -- the MicroPower/Pascal kernel, OTS, or a MACRO-11 program -- PASDBG cancels scope.

SHOW SCOPE
Displays the current lexical context for symbol references, including program, module, and scope settings.

CANCEL SCOPE
Clears the current scope setting; program and module settings remain in effect. No scoped Pascal data or code references are allowed until a new scope is set.

The following example illustrates the relationship between the PROGRAM and SCOPE commands and shows the concept of lexical positions.

```
[SYSTEM(MICROPOWER)] PROGRAM OUTER;
VAR I : INTEGER;

  PROCEDURE INNER;
  VAR I : BOOLEAN;
  BEGIN {INNER}
    I := TRUE;
    END;

BEGIN {OUTER}
    INNER;
    I := 3;
END.
```

The example uses an integer variable called "I" and a Boolean variable called "I". To examine either variable, you must set lexical scope to the block of code where the variable is found as a local variable. For example, to examine the integer variable I, use the SET PROGRAM command to set scope to program OUTER. You can now examine the integer variable I, because the SET PROGRAM command automatically sets scope to the main block of the program -- in this case, OUTER -- and integer variable I is local to this block. Refer to the sample dialog below for an illustration.

To examine the Boolean variable I, you must use a more specific scope. With program set to OUTER, use the SET SCOPE command to specify the lexical path to the location where Boolean variable I is a local variable. In this case, scope is INNER because Boolean variable I is a local variable of the INNER procedure of program OUTER. Refer to the sample dialog below.

```
PASDBG>SET PROGRAM OUTER<RET>

PASDBG>EXAMINE I<RET>
013245: 3

PASDBG>SET SCOPE INNER<RET>

PASDBG>EXAMINE I<RET>
012425: TRUE
```

2.1.2.2 **PROCESS** - Every process has a stack -- an area in memory for maintaining the values of variables local to that process and the values of variables local to Pascal routines called by the process. Because one process can repeatedly initiate a second process that is capable of running independently of the first, thereby creating other stacks with the same variables but different values, PASDBG requires that you establish a process context -- "set process" -- before you access a process's local variables. With the PROCESS commands, you tell PASDBG exactly which process's variables you wish to access. (See also the SET PROGRAM command.)

Wherever the application system stops, PASDBG resets process to the currently running process, if any, and displays the new process setting.

SET PROCESS
Establishes a process context and, in a virtual system, sets mapping to the specified process. SET PROCESS makes it possible to access a process's stack and local variables.

SHOW PROCESS
Displays the currently set process, a process you specify, or all processes.

CANCEL PROCESS
Cancels the process set with SET PROCESS, cancels the lexical scope set with SET SCOPE, and performs a SET PROCESS to the program set with SET PROGRAM. If no SET PROGRAM was issued, physical addressing is set. After process has been canceled, variables local to the canceled process can be accessed only by physical address.

2.1.3 Controlling Program Execution

The following commands allow you to control the execution of your application program.

```
GO
SET STEP
SHOW STEP
STEP
CANCEL STEP
SET BREAK
SHOW BREAK
CANCEL BREAK
SET TRACE
SHOW TRACE
CANCEL TRACE
SET WATCH
SHOW WATCH
CANCEL WATCH
SHOW CALLS
INIT
```

2.1.3.1 Starting and Stopping Execution

- The GO and STEP commands are used to regulate manually the execution of your program.

When you issue the GO command, PASDBG instructs the application to restart from the point at which it was stopped. If the application has not yet run, the application starts at its starting address in the kernel. You must issue a GO command to start the application program initially when using the debugger. To restart an application from its starting address, use the INIT/RESTART command.

When you issue a GO command, PASDBG displays the following message:

```
[Target execution resumed - type <CR> to stop target]
```

To interrupt program execution, type a carriage return. PASDBG will respond by resetting scope to the currently running program, procedure, function, or process, reporting on the state of the application, and returning to PASDBG> prompt level.

The /EXIT qualifier to the GO command instructs PASDBG to start or continue program execution and then exit to the host monitor, leaving the application running.

The STEP command permits you to go through the execution of the application program step by step. You can step between Pascal program statements or PDP-11 instructions. You can also step into or skip over all subroutine calls, including calls to the MicroPower/Pascal object time system (OTS).

GO
Starts the application at its starting point in the kernel or restarts the application from the point at which it was stopped. After issuing a GO, type a carriage return to stop the application program. The /EXIT qualifier causes PASDBG to exit to the host monitor after (re)starting the application.

SET STEP
Sets the parameters for the STEP command. INTO directs the debugger to step into subroutine calls, OVER to skip over subroutine calls, INSTRUCTION to step by PDP-11 instruction increments, and STATEMENT to step by Pascal statement increments.

SHOW STEP
Lists the current step parameters.

STEP
Single-steps through program execution in instruction or statement increments. The /INSTRUCTION, /STATEMENT, /INTO, and /OVER qualifiers can be used to override the current step parameter settings.

CANCEL STEP
Cancels the step parameters set with SET STEP and resets the parameters to the system default -- STATEMENT, INTO.

2.1.3.2 **BREAK, TRACE, and WATCH** - The BREAK, TRACE, and WATCH commands permit you to set breakpoints, tracepoints, and watchpoints in your application program.

Breakpoints notify you and stop the application program whenever a specified instruction is about to execute.

Tracepoints notify you but do not stop the application program whenever a specified instruction is about to execute.

Watchpoints notify you and stop the application program whenever a specified location is modified.

When your program stops because it triggered a breakpoint or a watchpoint, the debugger prompts you for command input. You can continue program execution with a GO or a STEP command, or you can enter other commands before continuing execution. The breakpoint, tracepoint, or watchpoint remains set after it is triggered -- until you cancel it with a CANCEL command.

You can defer a breakpoint, tracepoint, or watchpoint by using the /AFTER switch to specify that the breakpoint, tracepoint, or watchpoint must be triggered a particular number of times before the debugger notifies you.

SET BREAK
Sets a breakpoint in the application program at the location you specify. A breakpoint stops the processor every time the instruction at the specified program location is about to execute.

SHOW BREAK
Lists all currently set breakpoints.

CANCEL BREAK
Removes one or all breakpoints.

SET TRACE
Sets a tracepoint in the application at the location you specify. A tracepoint notifies you every time the instruction at the specified program location is about to execute, then continues execution.

SHOW TRACE
Lists all currently set tracepoints.

CANCEL TRACE
Removes one or all tracepoints.

SET WATCH
Sets a watchpoint on the variable or location you specify. A watchpoint stops the processor every time the specified location is modified.

SHOW WATCH
Lists all currently set watchpoints.

CANCEL WATCH
Removes one or all watchpoints.

2.1.3.3 **Invocation Chain Display** - The SHOW CALLS command locates the chain of called routines -- procedures, functions, and processes -- in the currently running process. Many different procedures, functions, or processes can call a particular routine. SHOW CALLS retroactively traces and lists the chain of calls that resulted in execution of the currently running routine. Do not confuse the chain of calls listed by SHOW CALLS with lexical scope. You display the lexical scope of a procedure, function, or process with the SHOW SCOPE command.

SHOW CALLS
: Lists the chain of procedures, functions, or processes that invoked the currently running routine.

2.1.3.4 **Reinitializing and Restarting** - The INIT command reinitializes the debugger's internal database, canceling breakpoints, tracepoints, and watchpoints and resetting scope and step parameters.

PASDBG performs an implicit INIT at start-up time and when it executes a LOAD command.

The INIT/RESTART command performs an INIT and then reinitializes the application to its starting address in the kernel. You should reissue a LOAD command if you suspect that the application program code has been corrupted.

INIT
: Cancels breakpoints, tracepoints, watchpoints; resets scope, program\module, and process; and resets the default step parameters. The /RESTART command option reinitializes the application to its starting point.

2.1.4 Examining and Modifying Data

The following commands are used to examine and modify your application program and data.

```
EXAMINE
DEPOSIT
SHOW STRUCTURE
```

2.1.4.1 **Variable Modification and Display** - The EXAMINE and DEPOSIT commands let you observe or modify specified locations, registers, or Pascal variables in your application. You can also examine, but not modify, specified Pascal statements.

Use EXAMINE to determine the current values of data and code in your program.

For data, use the EXAMINE command to display the contents of a PDP-11 location or register or -- if program, scope, and process are set -- the contents of a Pascal variable. PASDBG displays a Pascal variable according to its data type as specified in the user program -- real-number variables are displayed as reals, characters are displayed as characters, and so forth.

For code, use the EXAMINE command to display instructions in PDP-11 format or to "disassemble" Pascal statements into the equivalent PDP-11 instructions.

Use DEPOSIT to change the contents of a memory location, a register, or a Pascal variable.

EXAMINE
Displays the contents of a location, a register, a statement, or a variable -- or a range of locations, registers, or statements.

DEPOSIT
Replaces the contents of a memory location, a register, or a Pascal variable.

2.1.4.2 Structure Display

- The SHOW STRUCTURE command has two functions. If you specify a variable or a field name, SHOW STRUCTURE lists the type of that variable or field and the names and types of any subfields in the variable or field. If you specify a Pascal routine name -- a procedure, process, program, or function name -- SHOW STRUCTURE lists the type of that routine, the names and types of any lexically subordinate routines, and the types of all local variables.

SHOW STRUCTURE
Displays the structure of a Pascal variable, function, procedure, process, or program.

2.1.5 General Debugging Aids

This section lists general-purpose debugger commands that do not fit into any of the categories covered in Sections 2.1.1 through 2.1.4. See also Section 2.2.4.

@
Invokes an indirect command file. PASDBG displays the commands from the file and the debugger response, if any, one command at a time.

LOG
Copies the debugging session dialog or, alternatively, only the commands you issued to PASDBG into a log file on the host system. You can use LOG to generate indirect command files.

CLOSE
Closes an open log file.

EXIT
Terminates execution of PASDBG and returns control to the host operating system. The application program does not halt but is left stopped. (To exit and leave the application running, use the GO/EXIT command.)

HALT
Forces the application program to stop. This command is not usually necessary, since <RET> after a GO command will stop the application program.

CTRL/C

If the application is executing, stops the application system and returns to PASDBG> prompt level. If a PASDBG command or a command procedure is executing, CTRL/C aborts the command or command procedure and returns to PASDBG> prompt level. If PASDBG has entered ODT mode (see the SET ODT command in Chapter 3), CTRL/C returns to PASDBG> prompt level. If you are at PASDBG> prompt level on an RT-11 host, CTRL/C cancels PASDBG and returns to the RT-11 monitor. On an RSX-11M/M-PLUS host, typing two CTRL/Cs aborts the debugger and returns control to the RSX-11M/M-PLUS operating system.

CTRL/Y

On a VAX/VMS host system, interrupts PASDBG execution and returns to the VMS operating system.

CTRL/O

Supresses screen output until the current PASDBG command completes or until you type another CTRL/O.

HELP

Provides an on-line HELP facility that provides information about PASDBG commands and qualifiers.

SPAWN

On an RSX-11M/M-PLUS or VAX/VMS host system, allows you to execute a single-line host command without exiting PASDBG.

2.2 REAL-TIME DEBUGGING

This section describes the PASDBG commands that, in conjunction with the general-purpose debugging commands described in Section 2.1, allow you to debug real-time programs. These commands report on the "typed" data structures in your application -- process control blocks (PCBs), binary and counting semaphores, ring buffers, and packet queues -- display the kernel-maintained process state queues, and list the exception-handling process groups established in your application. With these commands, you can determine at successive stages of execution whether the state of the processes in your application conforms to your expectations.

The following commands are used to debug real-time process control structures in MicroPower/Pascal:

```
SHOW PCB
SHOW SEMAPHORE
SHOW RING BUFFER
SHOW PACKET QUEUE
SHOW NAMES
SHOW FREE STRUCTURES
SHOW FREE PACKETS
SHOW INACTIVE QUEUE
SHOW READY/ACTIVE QUEUE
SHOW READY/SUSPENDED QUEUE
SHOW RUN QUEUE
SHOW EXCEPTION GROUPS
SHOW EXCEPTION
```

2.2.1 Typed Data Structure Display

Typed data structures are system data structures that are created and deleted by processes, via primitive operations. The SHOW PCB, SHOW SEMAPHORE, SHOW RING BUFFER, and SHOW PACKET QUEUE commands display information about the following typed data structures: the process control block (PCB), the binary semaphore, the counting semaphore, the ring buffer, and the packet queue. (See Chapter 2 of the *MicroPower/Pascal Runtime Services Manual* for a detailed description of typed data structures.)

The SHOW PCB command displays information contained in the process control block for the process you specify. This information describes the relation of the process to the MicroPower/Pascal system environment.

Use the SHOW PCB command to determine the state of one or more processes in an application. By issuing a SHOW PCB command for all processes, you can determine the current status of the entire application program.

The SHOW PCB command displays the process's name, serial number, state, priority, suspend count, termination address, exception class, stack limits, context switch options, and PCB address. SHOW PCB also displays the exceptions handled by the process and the exception-handler address for the process. If the process is blocked, SHOW PCB displays the name, address, and serial number of the control structure on which the process has blocked.

Note that depending on the current state and options used when you created the process, PASDBG may not display some of the SHOW PCB information.

The SHOW SEMAPHORE, SHOW RING BUFFER, and SHOW PACKET QUEUE commands indicate the state of the control variable associated with the specified structure and list any processes currently waiting for a change in the variable. SHOW SEMAPHORE displays the value of the semaphore, and SHOW RING BUFFER displays the number of bytes currently in use in the buffer. Processes waiting on a semaphore or a ring buffer are "blocked" and are held on the packet queue. To run, these processes require a change to the control variable from another process or interrupt service routine (ISR). Therefore, to determine all the processes in a blocked condition, you must use the SHOW SEMAPHORE, SHOW RING BUFFER, and SHOW PACKET QUEUE commands to examine all the existing control variables. Alternatively, you can use SHOW PROCESS/ALL or SHOW PCB to examine all the existing processes.

A packet queue is a MicroPower/Pascal control structure that receives packets of information from processes that signal it. The information is delivered to the process waiting on the packet queue. Packet queues can be priority or FIFO (first in, first out) ordered for both packets and processes on the queue. With FIFO ordering, the processes and/or packets leave the queue in the order in which they entered. With priority ordering, the highest priority process and/or packet leaves the queue first.

Although named a queue, a packet queue functions in the same manner as a semaphore. Other queues in MicroPower/Pascal are process state queues created by the kernel during the program's initialization; only one copy of each state queue exists. Semaphores and packet queues are created by user program commands, and many of each type may exist.

The SHOW NAMES, SHOW FREE STRUCTURES, and SHOW FREE PACKETS commands provide information about kernel data structures in your application. SHOW NAMES lists all the current named data structures. SHOW FREE STRUCTURES displays the amount of memory available for new kernel structures. SHOW FREE PACKETS displays the number of unused packets in the kernel packet pool.

SHOW PCB
Displays information stored in the process control block for the process you specify. Depending on the argument you give, SHOW PCB lists information about the currently set process or a particular process.

SHOW SEMAPHORE
Displays information about a semaphore, including the semaphore type -- binary or counting -- a list of processes blocked on the semaphore, the ordering of the waiting-process list (FIFO or priority ordered), and the state of the semaphore variable.

SHOW RING BUFFER
Displays information about a ring buffer, including its size, a list of processes waiting for characters from the ring buffer, a list of processes waiting for space to enter characters into the ring buffer, the ordering of the waiting-process lists (FIFO or priority ordered), and the number of bytes in the buffer.

SHOW PACKET QUEUE
Displays information about a packet queue, including a list of processes blocked on the packet queue, a list of the packets waiting on the queue and their priorities, and the ordering of the waiting-process list and the packet list (FIFO or priority ordered).

SHOW NAMES
Displays a list of all the currently named kernel structures, including PCBs, semaphores, ring buffers, and packet queues.

SHOW FREE STRUCTURES
Displays the amount of memory that is left in the kernel free-memory pool for new kernel structures. You can use this command to keep track of the kernel pool as structures are created and destroyed.

SHOW FREE PACKETS
Displays the number of unused packets in the kernel packet pool. You can use this command to determine whether your configuration file allocated enough packets.

2.2.2 State Queue Display

The SHOW RUN QUEUE, SHOW READY/ACTIVE QUEUE, SHOW READY/SUSPENDED QUEUE, and SHOW INACTIVE QUEUE commands display information about processes on the kernel-maintained state queues. The run queue contains the active process. The ready/active queue contains processes that are waiting to enter the run queue; they are there because their priority is not higher than the priority of the running process. Processes on the ready/suspended queue have been suspended either by themselves or by another process; they will enter the ready/active queue when another process issues them a RESUME. The inactive queue contains processes that have aborted with fatal exceptions -- exceptions for which no exception handling was established.

There are no commands for changing the contents of a queue.

SHOW RUN QUEUE
Lists the currently running process. Only one process can be in this queue at any one time.

SHOW READY/ACTIVE QUEUE
Lists the processes that are waiting to enter the run queue.

SHOW READY/SUSPENDED QUEUE
Lists the processes that have been suspended by themselves or by another process. These processes are waiting for a RESUME command from another process.

SHOW INACTIVE QUEUE
Lists the processes that have aborted with fatal exceptions.

2.2.3 Exception-Handling Display

The SHOW EXCEPTION GROUPS command displays current information about exception handling in your program. Refer to Section 2.3.7 and Chapter 3 for more information on exception handling with PASDBG.

The SHOW EXCEPTION command displays the last exception message generated by the application and sent to the host. You cannot use this command after execution has been restarted.

SHOW EXCEPTION GROUPS
Displays a list of the process groups with established exception-handling processes. The command also lists the types of exceptions that are handled for each process group and the packet queue that is signaled when one of the listed exceptions occurs in one of the listed process groups.

SHOW EXCEPTION
Displays the last exception message generated by the target and sent to the host system -- if no GO command was issued in the interim.

2.2.4 Real-Time Debugging Aids

This section lists real-time debugging commands that do not fit into any of the categories covered in Sections 2.2.1 through 2.2.3. See also Section 2.1.5.

SHOW TARGET
Displays the current state of the application system, including the physical location at which the application system has stopped, the instruction it is about to execute, the currently running process, and whether or not the program is using memory management on the target.

SET ODT
Forces the application system to halt and enter Micro-ODT mode. All further entries are interpreted as commands to the target computer's Micro-ODT (refer to Section 2.3.6).

2.3 DEBUGGING HINTS

This section presents practical hints and suggestions to keep in mind as you use PASDBG.

2.3.1 Real-Time Errors

It can often be difficult to distinguish between time-independent and real-time errors, especially when the error does not show an obvious pattern. This section presents some ideas on how you can use PASDBG to locate and analyze real-time errors.

To isolate these errors, begin by examining the state queues -- for example, run, ready/active, ready/suspend -- and the control mechanisms -- for example, ring buffers and semaphores. Look for an impossible combination of events based on the expected process synchronization.

Another clue can often be found by varying the code execution speed. Set the T-bit ON all the time -- with SET WATCH or STEP -- run the program, and see if the problem disappears. Check the states of all the processes when, or just before, the problem occurs. It may be helpful to set a watchpoint on the kernel variable $RUN; this causes the application to stop whenever the running process changes.

2.3.2 Critical Sections

The critical-section problem is one of the basic errors that occurs in real-time programming environments. Critical-section problems are eliminated by exclusive use of MicroPower/Pascal control structures. Figure 2-1 gives an example of a critical-section problem.

```
[ SYSTEM(MICROPOWER) ] PROGRAM FUBAR;
VAR A : INTEGER;
    INUSE : BOOLEAN;

[PRIORITY(100)] PROCESS ONE;
BEGIN
        WHILE INUSE DO;
        INUSE := TRUE;
        A := 1;
        INUSE := FALSE;
END;

[PRIORITY(100)] PROCESS TWO;
BEGIN
        WHILE INUSE DO;
        INUSE := TRUE;
        A := 2;
        INUSE := FALSE;
END;

BEGIN           {MAIN PROGRAM}
        .
        .
        .
END.
```

Figure 2-1 Program Example Using Flags for Process Control

Assume that the two processes in the program example are set to run depending on the receipt of different interrupts by the system. The sequence of events that causes the error is as follows:

1. Process ONE executes.
2. After executing WHILE INUSE and before executing INUSE := TRUE, process ONE is interrupted.
3. Process TWO now starts running in response to the interrupt.
4. Process TWO executes into its main body of code (near A := 2).
5. Another interrupt or significant event occurs.
6. Process ONE resumes execution.
7. Depending on whether or not process TWO completed statement A := 2, the eventual state of A can be either 1 or 2 because process TWO will eventually resume execution at the point where it was interrupted. Thus, the state of variable A is indeterminate.

The key to avoiding critical-section problems is to make sure that the test and the set of each process -- that is, WHILE INUSE DO and INUSE := TRUE, in this case -- cannot be interrupted. You could attempt to solve this problem by combining the two statements into one. Even if the two statements could be combined, however, the compiler may expand the statement into several machine-language instructions, and the problem could reappear. Rather, the best solution is always to use MicroPower/Pascal control structures such as semaphores, because they cannot be interrupted. Interrupts that change the state of MicroPower/Pascal's access to the control structures are queued; no semaphore can be changed until any previously started primitive operation is complete. Figure 2-2 shows that the error disappears if the program is written using a binary semaphore, the most simple MicroPower/Pascal control structure.

```
[ SYSTEM(MICROPOWER) ] PROGRAM FUBAR;
VAR A : INTEGER;
    INUSE : SEMAPHORE_DESC;
    D : BOOLEAN;

[PRIORITY(100)] PROCESS ONE;
BEGIN
        WAIT(DESC := INUSE);
        A := 1;
        SIGNAL(DESC := INUSE);
END;

[PRIORITY(100)] PROCESS TWO;
BEGIN
        WAIT(DESC := INUSE);
        A := 2;
        SIGNAL(DESC := INUSE);
END;

BEGIN          {MAIN PROGRAM}
        D := CREATE_BINARY_SEMAPHORE(DESC := INUSE);
        .
        .
        .
END.
```

Figure 2-2 Program Example Using Semaphores for Process Control

2.3.3 **Race Conditions**

Race conditions are another cause of real-time errors. In a race, two program elements attempt to use the same resource or device simultaneously. MicroPower/Pascal control structures are designed to avoid these race conditions and to make those that do occur easy to find and correct.

Four examples of race conditions are described below.

1. **Deadlocks** Two processes require the use of the same two resources. One process locks into the first resource, and another process locks into the second resource. Each process then attempts to access the other resource without giving up the resource it already owns. The result is that neither process can continue operating.

2. **Data Juxtaposition** Two processes request data from the same device without identifying themselves as the source of the request. The data, therefore, can be returned to the wrong process. For example, process A submits a disk read request and then stops. Process B then starts and submits its own read request, which is queued behind A's request. Process B sees the done flag from the disk and reads the information gathered for process A. Process B then does something indeterminate because it has the wrong information; if process A then executes, it would make the same mistake as B did, because it would read the information gathered for process B.

3. **Unprotected Protection Controls** Flags can be set to protect critical sections of code, often causing the flags to become critical sections. If a process does not set the proper types of flags in the proper order, the process may be interrupted by false signals caused by the improperly set flags. MicroPower/Pascal process control variables are designed to avoid this type of problem.

4. **Insufficient Processing Time** (This is a race condition in reverse.) The following example describes an insufficient processing time problem caused when an external device streams a high rate of interrupts to a processor. The processor responds to the incoming interrupts with an interrupt service routine (ISR). If the ISR does not disable further interrupts then the ISR cannot complete execution before the next interrupt arrives. If the ISR disables interrupts then interrupts are lost. In the first case, the system is likely to crash because of the excessive number of ISRs queued to handle the interrupts or because one interrupt scrambles another interrupt's data. In the second case, data is lost because the flow of interrupts is cut off.

2.3.4 **STEP and WATCH Commands**

The STEP and WATCH commands run slowly. Use them only when appropriate.

2.3.5 SET WATCH with Local Variables

To set a watchpoint on a local variable in a process other than the static process, you must SET PROCESS to the process that contains the local variable. However, if the process has not been called or the routine containing the variable has not executed, the local variable does not yet exist. Until the process is called and the routine executes, you cannot set a watch on the variable; nor can you examine it.

Furthermore, if you set watch on a local variable and the process in which you set the watchpoint terminates, the process's stack space for local variables is released, but the watchpoint is still set on the old stack location. If another copy of the process is then called, new copies of the variables are created on the stack. A watchpoint set on a variable in the previous invocation is now set on an unknown variable in the new invocation -- assuming the stack was allocated in the same area. The watchpoint, under these circumstances, first points to the variable, then points to nothing, then points to something unknown.

To use the SET WATCH command on a local variable, do the following:

1. Set a breakpoint on the first line of code in the routine.
2. When that breakpoint is triggered, set your watchpoint.
3. Before restarting, set a breakpoint on any termination point in the process.
4. If you wish, cancel the breakpoint on the process entry point.
5. Continue debugging.
6. When a termination point breakpoint triggers, cancel all the watchpoints for the routine.
7. If you canceled the entry point breakpoint (step 4) and you wish to set the watch again if the process is restarted, repeat from step 1.

You can set breakpoints on a process that has not yet been created. In fact, you can perform all manipulations involving code locations, because the code for the process always physically exists. Use the SET PROGRAM and the SET SCOPE commands -- not SET PROCESS -- to gain access to the static components of the not yet created process.

2.3.6 Micro-ODT (uODT) Mode

If the target system halts while you are using PASDBG, the system enters Micro-ODT (uODT) mode. PASDBG senses this transition and displays the following message:

```
?PASDBG-E-TARHALT, Target halted - PASDBG in uODT mode
@
```

NOTE

The @ symbol displayed is the Micro-ODT prompt. Note that this is the only situation in which PASDBG prints the @ symbol. All other occurrences of the @ symbol are limited to commands you type.

The @ symbol and all subsequent Micro-ODT dialog are displayed in reverse video. If you are using a terminal that does not respond to VT100 escape sequences for reversing the video, extraneous characters preceding the @ symbol will be displayed.

You can now use Micro-ODT as if your terminal were directly connected to the application so that you can obtain additional debugging information. All commands typed at the host system console terminal are sent to the target system as Micro-ODT commands, and the target system response is displayed on the terminal. See the Microcomputer Processor Handbook for instructions on how to use Micro-ODT.

To exit from Micro-ODT dialog, type CTRL/C or CTRL/A. Control returns to PASDBG, which will then respond to user commands; however, the target system remains halted and in Micro-ODT mode.

The target system Micro-ODT can also be entered by typing the SET ODT command to PASDBG. See the SET ODT section of Chapter 3 for details.

2.3.7 Fatal Exceptions

If a fatal exception is raised by the application program while you are debugging, the offending process is aborted and the exception is reported to PASDBG. If the exception was raised by the kernel, PASDBG prints:

```
FATAL KERNEL TRAP address
```

In this message, "address" is the address of the instruction after the one that caused the fatal trap.

If the exception was raised by main-line Pascal code, by the Pascal object-time system (OTS), or by any other nonkernel code, PASDBG reports the class and subcode of the exception, the physical and virtual address of the instruction that caused the exception, and the value of the PC after the exception was detected. If the exception was raised by Pascal code or the OTS, PASDBG also reports the number of the statement that caused the exception.

Any attempt to continue program execution after a fatal exception has been raised may produce unpredictable results unless the application was specifically designed to allow the offending process to terminate without disruption of the application.

Exception codes are listed in the MicroPower/Pascal messages manual for your host system. Exception handling is discussed in detail in the MicroPower/Pascal Runtime Services Manual, Chapter 7, and in the MicroPower/Pascal Language Guide, Chapter 17.

2.3.8 Debugging FALCON or FALCON-PLUS Applications

PASDBG is used to debug SBC-11/21 applications in essentially the same manner as with any other MicroPower/Pascal application. However, because the SBC-11/21 does not provide Micro-ODT, the console emulator required by PASDBG, Macro-ODT -- a Micro-ODT emulator contained on the KXT11-A2 PROM set -- must be added to the SBC-11/21 board. The Macro-ODT program performs the following functions for the SBC-11/21 processor: power-up, bootstrap, down-line load, halt and break intercepts, trap-to-4 emulation, and power-up diagnostics.

All SBC-11/21 configurations can be used with PASDBG. Some configurations, however, require that the SBC-11/21 be turned off, restarted, and initialized by Macro-ODT each time the application is loaded. This "cold-start" procedure is necessary for those configurations that either use nonstandard BREAK jumpers or specify a SYSHALT parameter other than ODTROM in the configuration file. In these cases, the break sent by PASDBG will cause the SBC-11/21 to hang rather than to enter Macro-ODT. Otherwise, if you specify SYSHALT+ODTROM, the application program can be reloaded without repeating the cold start.

All PASDBG commands function normally with the SBC-11/21. The application cannot, however, service any interrupt signals while WATCH or STEP is set, because the SBC-11/21 processor does not allow interrupts to be processed when the T-bit is set on, and PASDBG uses the T-bit to perform the WATCH and STEP functions. Therefore, you must be careful of where and how WATCH and STEP are used with SBC-11/21 applications, especially when you are debugging time-dependent programs.

2.3.9 Debugging KXT11-C Arbiter/Slave Protocol Transactions

The MicroPower/Pascal kernel defines two debug locations -- $KXTQW and $KXTQR -- that can be used to debug KXT11-C arbiter/slave protocol transactions. (The KXT11-C arbiter/slave protocol is implemented via the KX handler running on the arbiter system and the KK handler running on the KXT11-C peripheral processor.)

The kernel-defined debug location $KXTQW is called by the KK handler just before data from the arbiter is transferred from a channel into a KXT11-C buffer -- a KX handler write operation.

The kernel-defined debug location $KXTQR is called by the KK handler after data has been transferred from a KXT11-C buffer to a channel but before the LSI-11 bus is interrupted -- a KX handler read operation.

You can use the PASDBG SET BREAK command to set a breakpoint on either or both of the debug locations. Doing so allows you to examine a segment of a message while a read/write operation is suspended in midexecution.

CHAPTER 3

COMMAND REFERENCE

This chapter provides information about debugger command format, parameters, and qualifiers for quick reference. Sections are ordered alphabetically by command name.

NOTE

> In this chapter, underscores (___) indicate the minimum acceptable abbreviations for debugger commands and qualifiers. For example, the command SHOW PCB appears as SHOW PCB. Other notational conventions used in this chapter are listed in the Preface.

The following operator symbols are used in debugger commands:

Symbol	**Description**
@	Indirect file-specification prefix
%	Radix keyword prefix
/	Command modifier prefix
\	Scope modifier prefix
!	Comment prefix
,	Parameter list separator
<CR>	End of line delimiter
..	Address range specification
'<filespec>'	Literal file specification
#	Breakpoint ordinal number prefix

@

3.1 @

PASDBG accepts commands from an indirect command file. To invoke an indirect command file, type @ followed by the file name. PASDBG displays the commands from the file and the debugger's response, if any, one command at a time.

Note that the LOG command can be used to generate command files for use with @.

NOTE

To exit from an indirect file before end-of-file has been reached, type CTRL/C.

Syntax

```
@[dev:]name[.ext]
```

Command Parameters

dev
: The name of the host-system device containing the indirect file.

name
: The indirect file's name.

ext
: The indirect file's extension. The default extension is .COM.

Example

```
PASDBG>@setup<RET>    !execute commands from setup.com
```

NOTE

Do not confuse the operator symbol @ with the prompt @. The prompt @, shown in reverse video, indicates that PASDBG has entered Micro-ODT mode. If the application system crashes, PASDBG enters Micro-ODT mode and issues the statement:

```
%PASDBG-E-TARHALT Target halted -
PASDBG in uODT mode
@
```

In Micro-ODT mode, PASDBG connects directly to the Micro-ODT facility on the application system. The PASDBG terminal now functions as if it were a terminal on the application system in Micro-ODT. For information on how to use Micro-ODT to gather additional debugging information, consult the _Microcomputer Processor Handbook_.

CANCEL BREAK

3.2 CANCEL BREAK

The CANCEL BREAK command removes either a particular breakpoint or all breakpoints.

You can cancel a particular breakpoint by specifying the breakpoint number, statement number, label number, or address of the breakpoint. If you are not sure of the ordinal number of a breakpoint, use the SHOW BREAK command to list all currently set breakpoints and their numbers.

If you specify the /ALL switch, PASDBG will cancel all breakpoints set.

To verify the cancellation of a breakpoint, use the SHOW BREAK command.

Syntax

```
CANCEL BREAK[/ALL]  [ { address-expression } ]
                    [ { #breakpoint-number } ]
```

Command Parameters

address-expression
An expression giving the location of the breakpoint to be canceled. Address-expressions may be one of the following:

- statement-number -- a Pascal statement number (unsigned integer) valid within the current scope.
- LABEL label-number -- a Pascal label number (unsigned integer) defined within the current scope, preceded by the keyword "LABEL" and a space.
- MACRO-global-name -- the name of a MACRO-11 global symbol defined in the current program. Dots (".") in the symbol name must be entered as underscores ("_"); for example, PC.LNK is entered as PC_LNK.
- @[radix]address -- a physical, kernel, or virtual address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal). The address is interpreted as a physical address if physical mapping is set, a kernel address if KERNEL mapping is set, or a virtual address in the currently set process if process mapping is set.

breakpoint-number
The ordinal number of the breakpoint to be canceled.

Command Qualifier

/ALL
Specifies that all breakpoints are to be canceled.

Examples

```
     PASDBG>cancel break 1<RET>    !Cancels breakpoint at statement
                                   !number 1 in the current scope.

     PASDBG>cancel break #2<RET>   !Cancels breakpoint number 2.

     PASDBG>cancel break LABEL 10<RET>    !Cancels the breakpoint set
                                          !at label 10.

     PASDBG>cancel break @%021354<RET>    !Cancels the breakpoint set
                                          !at octal location 021354.
```

3.3 CANCEL PROCESS

The CANCEL PROCESS command cancels the process selected with SET PROCESS, cancels the scope selected with SET SCOPE, and performs a SET PROCESS to the program set with SET PROGRAM. If no SET PROGRAM was issued, PASDBG sets physical addressing only.

The CANCEL PROCESS command has no arguments.

Syntax

```
CANCEL PROCESS
```

Example

```
PASDBG>show proc<RET>
 Process 14, (no name), PCB at KERNEL (022416)

PASDBG>cancel proc<RET>
 Current process set to:
 Process 3, name = 'PROCS1', PCB at KERNEL (021620)
 Scope set to:
 Program PROCS1, Module PROCS1
```

3.4 CANCEL SCOPE

The CANCEL SCOPE command revokes the current lexical scope setting. No scoped Pascal data or code references are allowed until a new scope is set. Program and module settings remain in effect after CANCEL SCOPE.

The command has no arguments.

Syntax

```
CANCEL SCOPE
```

Example

```
PASDBG>show scope<RET>
 Program PROCS2, Module PROCS2, Scope: P1

PASDBG>cancel scope<RET>

PASDBG>show scope<RET>
 Program PROCS2, Module PROCS2
```

CANCEL STEP

3.5 CANCEL STEP

The CANCEL STEP command restores the SET STEP default parameters: STATEMENT, INTO. The command has no arguments.

Syntax

```
CANCEL STEP
```

Example

```
PASDBG>show step<RET>

Step parameters:  instruction into

PASDBG>cancel step<RET>

PASDBG>show step<RET>

Step parameters:  statement into
```

3.6 CANCEL TRACE

The CANCEL TRACE command cancels either a single tracepoint or all currently set tracepoints.

To cancel a single tracepoint, you must specify the address of the tracepoint, its ordinal number, its label number, or its statement number. If you are not sure of the ordinal number of a tracepoint, use the SHOW TRACE command to list all currently set tracepoints and their numbers.

To cancel all current tracepoints, use the /ALL qualifier.

To verify the cancellation of a tracepoint, use the SHOW TRACE command.

Syntax

$$\underline{\text{CA}}\text{NCEL } \underline{\text{T}}\text{RACE [/}\underline{\text{AL}}\text{L]} \left[\left\{ \begin{array}{l} \text{address-expression} \\ \text{\#tracepoint-number} \end{array} \right\} \right]$$

Command Parameters

address-expression
: An expression giving the location of the tracepoint to be canceled. Address-expressions may be one of the following:

- statement-number -- a Pascal statement number (unsigned integer) valid within the current scope.
- LABEL label-number -- a Pascal label number (unsigned integer) defined within the current scope, preceded by the keyword "LABEL" and a space.
- MACRO-global-name -- the name of a MACRO-11 global symbol defined in the current program. Dots (".") in the symbol name must be entered as underscores ("_"); for example, PC.LNK is entered as PC_LNK.
- @[radix]address -- a physical, kernel, or virtual address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal). The address is interpreted as a physical address if physical mapping is set, a kernel address if KERNEL mapping is set, or a virtual address in the currently set process if process mapping is set.

tracepoint-number
: The ordinal number of the tracepoint to be canceled.

Command Qualifier

/ALL
: Specifies that all tracepoints are to be canceled.

Examples

```
PASDBG>cancel trace #2<RET>       !Cancels tracepoint number 2

PASDBG>cancel trace LABEL 2<RET> !Cancels the tracepoint set at
                                  !label 2
```

3.7 CANCEL WATCH

The CANCEL WATCH command removes either a particular watchpoint or all watchpoints.

You can cancel a particular watchpoint by specifying either the watchpoint number or address of the watchpoint. If you have two watchpoints set to the same location, you should use the ordinal numbers of the watchpoints to cancel one or both of them. If you are not sure of the ordinal number of a watchpoint, use the SHOW WATCH command to list all currently set watchpoints and their numbers.

If you specify the /ALL switch, PASDBG will cancel all watchpoints set.

To verify the cancellation of a watchpoint, use the SHOW WATCH command.

Syntax

$$\text{\underline{CA}NCEL \underline{W}ATCH[/\underline{AL}L]}\left[\begin{Bmatrix}\text{address-expression}\\ \text{\#watchpoint-number}\end{Bmatrix}\right]$$

Command Parameters

address-expression
An expression giving the location of the watchpoint to be canceled. Address-expressions may be one of the following:

- variable-name -- the name of a Pascal variable defined within the current scope. Structure names are prohibited. However, you can specify a record field or an array element if it is a simple type.
- MACRO-global-name -- the name of a MACRO-11 global symbol defined in the current program. Dots (".") in the symbol name must be entered as underscores ("_"); for example, PC.LNK is entered as PC_LNK.
- @[radix]address -- a physical, kernel, or virtual address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal). The address is interpreted as a physical address if physical mapping is set, a kernel address if KERNEL mapping is set, or a virtual address in the currently set process if process mapping is set.

watchpoint-number
The ordinal number of the watchpoint to be canceled.

Command Qualifier

/ALL
Specifies that all watchpoints are to be canceled.

Examples

```
PASDBG>cancel watch #1<RET>        !Cancels watchpoint number 1

PASDBG>cancel watch @%Ø21354<RET> !Cancels the watchpoint set
                                   !at octal location Ø21354
```

3.8 CLOSE

The CLOSE command closes an open log file. If no log file is open, the command has no effect.

Syntax

```
    CLOSE [LOG]
```

Command Parameter

LOG
    Has no effect; it is included for clarity.

3.9 CTRL/C

The CTRL/C command (hold down the CTRL key while typing C) can be used to stop the application program, to abort a PASDBG command or command procedure, to exit from Micro-ODT mode, or, on an RT-11 or RSX-11M/M-PLUS host, to exit from PASDBG.

If the application is running, CTRL/C stops the application and returns to PASDBG> prompt level.

If a PASDBG command or a command procedure is executing, CTRL/C aborts the command or command procedure and returns to PASDBG> prompt level.

If the debugger is in Micro-ODT mode (see the SET ODT command), CTRL/C terminates the ODT dialog and returns to PASDBG> prompt level.

If CTRL/C is typed at the PASDBG> prompt level on an RT-11 host, PASDBG is canceled, and control returns to the RT-11 monitor.

On an RSX-11M/M-PLUS host, typing two CTRL/Cs aborts the debugger and returns control to the RSX-11M/M-PLUS operating system.

Syntax

```
     CTRL/C
```

3.11 DEPOSIT

The DEPOSIT command changes the contents of a Pascal variable, a memory location, or a register.

To deposit into a Pascal variable, set program, process, and scope to the routine containing the variable. You can then modify the variable, referring to it by its Pascal name and specifying a value to replace its current value -- for example, dep r = 1.1.

In most instances, the type of the data you specify should match the type of the variable you are modifying. However, PASDBG permits you to deposit data into a variable without regard to type, so long as the data size is not larger than the variable size. The application program will then interpret the data as if it were of the type defined in the source code. For example, a DEPOSIT of the character 'A' into an integer variable would force the ASCII value of the character (65 decimal) into the variable.

You can deposit up to four bytes of character data or real data with one command. In the case of character data, you can deposit a string of up to four characters. Note, however, that since data size cannot exceed variable size, you cannot deposit a 4-character string into a Pascal character variable, but only into a Pascal real variable or a specified address. In the case of real data, you can deposit one real number. (All real numbers are four bytes long.) If you deposit a real number into a register, PASDBG deposits the two low bytes into the specified register and the two high bytes into the next higher register.

NOTE

For your protection, PASDBG does not permit deposits into R6 or R7 (the stack pointer and the program counter).

If you deposit into a PACKED variable, the debugger automatically packs the new data into the variable.

DEPOSIT/BYTE deposits a byte of data at a specified address. You must use the /BYTE option to deposit data into an odd-numbered address. DEPOSIT/BYTE can also be used to deposit three bytes of character data (see the examples below).

You cannot deposit into a local variable in a process or a procedure that has not yet been invoked. Nor can you deposit into a label or a statement.

Syntax

$$\underline{\text{D}}\text{EPOSIT[/}\underline{\text{BY}}\text{TE]}\left\{\begin{array}{c}\text{address-expression}\\ \text{register}\end{array}\right\}\text{ [:]= [radix]data}$$

Command Parameters

address-expression
An expression specifying a location to be modified. Address-expressions may be one of the following:

- variable-name -- the name of a Pascal variable defined within the current scope. Set names are prohibited.
- MACRO-global-name -- the name of a MACRO-11 global symbol defined in the current program. Dots (".") in the symbol name must be entered as underscores ("_"); for example, PC.LNK is entered as PC_LNK.
- @[radix]address -- a physical, kernel, or virtual address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal). The address is interpreted as a physical address if physical mapping is set, a kernel address if KERNEL mapping is set, or a virtual address in the currently set process if process mapping is set.

register
A PDP-11 instruction register specification (%RØ to %R5, %PS, or %R8).

radix
Any of the following: %D -- decimal; % or %O -- octal; %X -- hexadecimal. Radix can only be used with unsigned integer data.

data
One of the following value representations: an integer; a real number; a character string; or a symbolic value, such as TRUE/FALSE or an enumerated type constant. If you are depositing into a variable, the data size must not exceed the variable size.

Command Qualifier

/BYTE
Instructs PASDBG to deposit one byte of data at the specified address. Alternatively, if the data specified is a 3-byte character string, PASDBG will deposit three bytes of data at the specified address.

Examples

```
DEPOSIT a[2]:=2          !Deposits decimal 2 into Pascal
                         !variable a[2] (part of an array
                         !called a).

DEP DOOR = RED           !Deposits RED into a Pascal variable
                         !DOOR, which is an enumerated type
                         !(such as RED, GREEN, MAGENTA,
                         !CANDYSTRIPE)

DEP SWITCH := TRUE       !Deposits TRUE into the Boolean
                         !variable SWITCH

DEPOSIT @%1ØØ:=%1Ø       !Deposits octal 1Ø into octal location
                         !1ØØ (bytes 1ØØ and 1Ø1)
```

```
DEPOSIT/BYTE @%177501='A'   !Deposits the character A in octal
                            !byte 177501. Byte 177500 is not
                            !affected.

DEPOSIT @200:=1.7E-02       !Deposits decimal 1.7E-02 into decimal
                            !locations 200 and 202 (bytes 200,
                            !201, 202, and 203).

DEPOSIT %R2 = %23           !Deposits octal 23 into register 2.

DEPOSIT @%177500 ='ABC'     !Deposits four characters into
                            !consecutive memory locations starting
                            !with octal location 177500, zero
                            !filling byte 177503.

DEPOSIT/BYTE @%177600='ABC' !Deposits three characters into
                            !consecutive memory locations starting
                            !with octal location 177600, leaving
                            !byte 177603 untouched.

DEPOSIT A[1] = Blue         !Correct array form.  Deposits Blue
                            !into A[1] of array Colors.

DEPOSIT Employee.age:=30    !Deposits 30 into the field age of
                            !record Employee.
```

3.12 EXAMINE

The EXAMINE command displays, on the console terminal, the contents of a specified location, register, statement, or variable -- or a range of locations, registers, or statements -- in the target system.

When you use the EXAMINE command to look at a MicroPower/Pascal variable, PASDBG knows the type of the variable and displays the variable in the appropriate format. In multilevel variables, such as records or arrays, PASDBG displays all fields subordinate to the specified name.

You can use the EXAMINE command to disassemble a Pascal statement or range of statements. See the Examples section below.

You cannot examine a local variable in a procedure that has not yet invoked or a process that has not yet initialized. You also cannot examine the priority or other attribute of a process that has not yet initialized.

Syntax

$$\underline{E}\text{XAMINE[/qualifier[/qualifier]]} \begin{Bmatrix} \text{addr-expr[..addr-expr]} \\ \text{register[..register]} \end{Bmatrix}$$

Command Parameters

addr-expr
An expression specifying a location to be examined. This parameter may be one of the following:

- statement-number -- a Pascal statement number (unsigned integer) valid within the current scope.
- variable-name -- the name of a Pascal variable defined within the current scope. Set names are prohibited. Also, Pascal variable names cannot be used in range specifications.
- <u>LA</u>BEL label-number -- a Pascal label number (unsigned integer) defined within the current scope, preceded by the keyword "LABEL" and a space.
- MACRO-global-name -- the name of a MACRO-11 global symbol defined in the current program. Dots (".") in the symbol name must be entered as underscores ("_"); for example, PC.LNK is entered as PC_LNK.
- @[radix]address -- a physical, kernel, or virtual address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal). The address is interpreted as a physical address if physical mapping is set, a kernel address if KERNEL mapping is set, or a virtual address in the currently set process if process mapping is set.

register
    A PDP-11 instruction register specification (%RØ to %R7, %SP, %PC, %PS, or %R8).

Command Qualifiers

/WORD
    Displays the data in word format.

/BYTE
    Displays the data in byte format.

/ASCII
    Displays the word in ASCII character format.

/INSTRUCTION
    Displays the data in PDP-11 instruction format.

/RAD5Ø
    Displays the word in RAD5Ø character format.

/REAL
    Displays the data in floating point format.

/BINARY
    Displays data in base 2.

/DECIMAL
    Displays data in base 1Ø.

/HEXADECIMAL
    Displays data in base 16.

/OCTAL
    Displays data in base 8.

Examples

Assuming that you have set program and scope correctly, type the following to examine a variable called FOO:

```
PASDBG>examine foo<RET>
(Ø12345): 43
```

PASDBG displays the octal virtual address of the variable in parentheses and then displays the value of the variable. The data type of the variable determines the display format that PASDBG uses. PASDBG will not permit you to display the variable in a format that is not valid for the variable (such as INSTRUCTION). To display the variable in another format, specify the virtual address of the variable.

If you examine an array or record, PASDBG will display the individual elements and fields of the structure. For example:

```
PASDBG>examine/instr a<RET>    !display 4-element array
%PASDBG-W-Illegal use of mode switch -- ignored
A[1] (Ø4ØØØØ) : 1
A[2] (Ø4ØØØ2) : 2
A[3] (Ø4ØØØ4) : 3
A[4] (Ø4ØØØ6) : 4
```

You could then examine the physical locations 40000..40006 in any other mode, if desired. The warning indicates that PASDBG ignored the /INSTRUCTION switch.

If you examine a Pascal statement, the debugger displays the first PDP-11 instruction generated by the specified statement. To completely disassemble the statement, specify a range of statements. For example:

```
     EXAMINE 1..2
```

Every PDP-11 instruction generated by statement 1 in the current scope would be displayed, followed by the first PDP-11 instruction generated by statement 2. To disassemble statements 1 and 2 in the current scope, type:

```
     EXAMINE 1..3
```

You can examine a variable of an enumerated type such as:

```
     TYPE DOOR = (RED,BLUE,POLKA-DOT,CHARTREUSE);
     VAR KNOB :  DOOR;
```

PASDBG will display the variable according to the defined values. For example:

```
     PASDBG>EX KNOB<RET>
     (Ø4171Ø) : RED
```

Boolean variables are displayed as TRUE or FALSE.

You may specify at most two switches in an EXAMINE command. For example, to display octal locations 1ØØ to 11Ø of the current program as OCTAL BYTES, type:

```
     EXAMINE/BYTE/OCTAL @%1ØØ..@%11Ø
```

The % signs on the address numbers indicate that they have been entered as octal addresses. The /OCTAL switch asks PASDBG to output the contents of the locations as octal numbers.

To examine a register, specify %Rn, where R denotes a register, and n is the register number (from Ø to 7). 'SP' is an alternate name for R6, the stack pointer register; 'PC' is an alternate name for R7, the program counter register; and 'PS' and 'R8' are alternate names for the PSW (processor status word). For example, to display the contents of R4 in octal, type:

```
     EXAMINE/OCTAL %R4
```

To display in hexadecimal the contents of registers R2 through R6, inclusive, type:

```
     EXAMINE/HEX %R2..%SP
```

If you use /REAL with the EXAMINE command, two registers are displayed. The specified register becomes the low-order 16 bits of the number; the next higher register becomes the high-order 16 bits.

Mapped system users note: To access memory by its physical address, issue a SET PHYSICAL command first.

Macro users note: PASDBG displays the effective address for an instruction that references the PC. For example, PASDBG displays JMP 2ØØ(PC) as "JMP 3Ø4" if the instruction is at location 1ØØ(octal). The effective address is shown just as MACRO-11 assembled it, using the program counter and the length of the instruction to determine what the PC will be when that part of the instruction is executed.

EXIT

3.13 EXIT

The EXIT command terminates a PASDBG debugging session. It does not halt the application system but leaves it stopped. Control returns to the host monitor. The EXIT command has no arguments.

To exit PASDBG and leave the application running, use the GO/EXIT command.

Syntax

```
    EXIT
```

3.14 GO

The GO command starts or continues the execution of the program that you are debugging. The first GO command starts the program at the application program's starting point in the kernel. Thereafter, GO continues execution from the point at which that execution was stopped. After issuing GO, PASDBG does not permit any commands except a carriage return, which stops the application program. When the processor stops for any reason, such as a carriage return or the triggering of a breakpoint, PASDBG resets scope to the currently running program, procedure, function, or process (if any), resets process to the currently running process (if any), displays information on the state of the application, and returns to command mode.

GO/EXIT instructs the debugger to exit to the host monitor after starting or continuing program execution.

The GO command has no arguments.

Syntax

```
GO[/EXIT]
```

Command Qualifier

/EXIT
: Instructs PASDBG to exit after starting or continuing program execution.

Example

```
PASDBG>go<RET>

 [Target execution resumed -- type <CR> to stop target]
<RET>
 Target stopped at physical (00037204), virtual (037204) : MOV
 R3,-(SP)
 Executing non-Pascal code
 Process  3, name = 'PROCS2', PCB at KERNEL (021626)
```

3.15 HALT

The HALT command stops the application system. When the program is stopped, PASDBG displays information as if a breakpoint had occurred.

In most cases, typing HALT is not necessary, because PASDBG treats a carriage return as a HALT command when the application system is running.

The HALT command has no arguments.

Syntax

```
     HALT
```

or

```
     CTRL/C
```

or

```
     <RET> (after you issued a GO command)
```

Example

```
     PASDBG>halt<RET>

      Target stopped at physical (00035002), virtual (035002) : BR
      34706
      Executing non-Pascal code
      Process  12, (no name), PCB at KERNEL (022272)
```

3.16 HELP

PASDBG provides an on-line HELP facility for all PASDBG commands. For a list of commands, type HELP. For more information on a specific command, type HELP command. PASDBG may notify you that more information is available on a particular keyword -- a parameter or qualifier -- used with the command you specified. To get the information, type HELP command keyword.

Syntax

```
HELP [command [keyword]]
```

Command Parameters

(no arguments)
: If no argument is given, PASDBG lists the commands you can specify.

command
: The name of a debugger command with which you need help.

keyword
: A parameter or a qualifier about which you want further information.

Example

```
PASDBG>help close<RET>

HELP

  CLOSE

    "CLOSE"

    Closes a log file opened with the LOG command.
```

INIT

3.17 INIT

The INIT command reinitializes the debugger's internal data base. INIT cancels all breakpoints, tracepoints, and watchpoints; resets scope, program\module, and process; resets the step increments to STATEMENT and INTO; and stops the application system, if necessary. After reinitializing, PASDBG displays information on the state of the application system and returns to command mode.

PASDBG performs an implicit INIT at start-up time and when it executes a LOAD command.

If you specify the RESTART switch on INIT, PASDBG reinitializes the application to its starting point and returns to command mode.

Syntax

```
INIT[/RESTART]
```

Command Qualifier

/RESTART
: Causes PASDBG to reinitialize the application to its starting point.

NOTE

> If any of the application code becomes corrupted, using INIT/RESTART will have unpredictable results.
>
> Target system RAM is not zeroed on INIT/RESTART. The LOAD command must be used for RAM to be zeroed.

Examples

```
PASDBG>go<RET>
 [Target execution resumed - type <CR> to stop target]
 ** BREAKPOINT #Ø  '1 PROCS2'

 Target stopped at physical (ØØØ36654), virtual (Ø36654) : MOV
 @#43516,-(SP)
 In statement 1 + Ø in
 Program PROCS2, Module PROCS2, Scope: PROCS2
 Process 3, name = 'PROCS2', PCB at KERNEL (Ø21626)

PASDBG>init<RET>

 Target stopped at physical (ØØØ36654), virtual (Ø36654) : MOV
 @#43516,-(SP)
 In statement 1 + Ø in
 Program PROCS2, Module PROCS2, Scope: PROCS2
 Process 3, name = 'PROCS2', PCB at KERNEL (Ø21626)
```

```
PASDBG>show break<RET>
;PASDBG-I-NONESET, None set

PASDBG>init/rest<RET>

 Target stopped at physical (00001534), virtual (001534) : JMP
 @#5656
 Executing KERNEL code
 No process set, KERNEL mapping in effect
```

3.18 LOAD

The LOAD command loads files into the target system and/or into PASDBG.

LOAD/TARGET down-line loads a copy of the application memory image (.MIM) file into the target system.

LOAD/SYMBOL loads a copy of the application program's symbol table (.DBG) file into PASDBG on the host. You must issue this command to access symbols with PASDBG. Also, commands that require access to kernel symbols, such as SHOW RUN QUEUE, do not work without LOAD/SYMBOL.

LOAD -- with no option and no file extension specified -- directs PASDBG to perform both a LOAD/TARGET and a LOAD/SYMBOL.

LOAD/EXIT down-line loads a copy of the application .MIM file into the target system, starts the application, and then exits from PASDBG to the host operating system. You use the LOAD/EXIT command to down-line load an application built without debug support. The application then executes on the target system independently of PASDBG. (For RT users, the LOAD/EXIT command provides an alternative to the MicroPower/Pascal-RT DLLOAD utility, which is described in the MicroPower/Pascal-RT System User's Guide.)

Note that a down-line load may take several minutes with PASDBG.

Syntax

```
     LOAD[/qualifier] [dev:]name[.ext]
```

Command Parameters

dev
: An optional host-system device name.

name
: The file name.

ext
: An optional file name extension. If you do not specify an extension, the default is .MIM for LOAD/TARGET and .DBG for LOAD/SYMBOL. If you specify an extension other than .MIM or .DBG, you must specify /TARGET, /EXIT, or /SYMBOL.

Command Qualifiers

/TARGET
: Instructs PASDBG to down-line load a specified application memory image file into the target system.

/SYMBOL
: Instructs PASDBG to load the symbol table file into its memory (necessary for symbolic debugging and for use of kernel-specific commands).

(no qualifier)
    Instructs PASDBG to perform a LOAD/TARGET, a LOAD/SYMBOL, or both, depending on the file extension given (.MIM, .DBG, or none specified, respectively).

/EXIT
    Instructs PASDBG to down-line load a specified application memory image file into the target system, to start the application, and then to exit. (Control is returned to the operating system on the host.) This option is intended for application programs built without debugger support.

Example

```
PASDBG>load procs2<RET>
;PASDBG-I-BOTWARN, Starting primary boot load, please wait...
;PASDBG-I-BOTLD, Primary boot loaded, getting closer...

 Target stopped at physical (00001234), virtual (001534) : JMP
 @#5656
 Executing KERNEL code
 No process set, KERNEL mapping in effect
```

This example down-line loads the file PROCS2.MIM into the target and loads the file PROCS2.DBG into PASDBG.

3.19 LOG

The LOG command opens a log file on a host-system disk. The debugger then writes to that file either all text appearing on the screen or only the commands you issue.

You can use this command to create a file of frequently used commands that you invoke with the @ command. Or you can save a copy of the debugger dialog for analysis later.

PASDBG writes only your commands to the log file if the extension on the file is .COM. The default log file extension is .LOG.

Syntax

$$\underline{\text{LOG}}\ \text{[dev:]name}\left[\begin{Bmatrix}\text{.ext}\\ \text{.COM}\end{Bmatrix}\right]$$

Command Parameters

dev
: The name of the device that will contain the log file.

name
: The log file's name.

ext
: The log file's extension. The default extension is .LOG.

.COM
: The command file extension. This instructs PASDBG to write to the log file only the commands you issue.

Example

```
PASDBG>log setup.com<RET>    !build a command file
```

3.20 SET BREAK

The SET BREAK command establishes a breakpoint at a specified address. A breakpoint stops execution of your application program when the instruction at the specified address is about to execute. When a breakpoint is triggered, PASDBG stops the application program, notifies the user, and returns to command mode.

You can assign a message to the breakpoint. PASDBG displays the message when the breakpoint is triggered. If you set the breakpoint on a Pascal statement number and do not assign a message, PASDBG assigns the statement number and the procedure name as the message. If you set the breakpoint on a Pascal label and do not assign a message, PASDBG assigns "LABEL n procedure name" as the message (n is the label number). PASDBG truncates messages that are longer than 18 characters.

When you set a breakpoint in a procedure, the breakpoint triggers each time the procedure is called, regardless of the number of calls. When you set a breakpoint in a process, the breakpoint triggers on all invocations of the process unless the /PROCESS switch is used to specify a particular process.

You cannot set more than eight breakpoints and tracepoints.

Syntax

```
     SET BREAK[/qualifier...] address-expression ['message']
```

Command Parameters

address-expression
An expression giving the location at which the breakpoint is to be set. Address-expressions may be one of the following:

- statement-number -- a Pascal statement number (unsigned integer) valid within the current scope.
- LABEL label-number -- a Pascal label number (unsigned integer) defined within the current scope, preceded by the keyword "LABEL" and a space.
- MACRO-global-name -- the name of a MACRO-11 global symbol defined in the current program. Dots (".") in the symbol name must be entered as underscores ("_"); for example, PC.LNK is entered as PC_LNK.
- @[radix]address -- a physical, kernel, or virtual address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal). The address is interpreted as a physical address if physical mapping is set, a kernel address if KERNEL mapping is set, or a virtual address in the currently set process if process mapping is set.

message
A message to be displayed when the breakpoint is triggered.

Command Qualifiers

/AFTER:integer

Sets a counter on the breakpoint, causing the breakpoint to trigger only after the application program has executed the instruction the number of times specified by integer. The default count is 1. After the breakpoint is triggered, the count is reset to Ø but the qualifier remains in effect. Thus, AFTER:2 causes the breakpoint to trigger every other time it is reached.

/PROCESS:process-id

Causes the breakpoint to trigger only if the selected process executes the instruction. The specified process must currently exist. Process-id, which selects the process, may be one of the following:

- process-descriptor -- a Pascal process-descriptor variable name.
- 'process-name' -- a runtime process name, enclosed in quotes.
- serial-number -- a runtime process serial number (unsigned integer).
- @[radix]PCB-address -- a process control block (PCB) address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal).

Example

```
PASDBG>set scope baker<RET>    !set break on baker, label 1Ø

PASDBG>set break LABEL 1Ø<RET>

PASDBG>go<RET>
 [Target execution resumed - type <CR> to stop target]
 ** BREAKPOINT #Ø 'LABEL 1Ø BAKER'

 Target stopped at physical (ØØØ5307Ø), virtual (Ø5307Ø) : MOV
 #2,@#533Ø4
 In statement 3 + Ø in
 Program ZEPHOD, Module ZEPHOD, Scope: BAKER
 Process 6, (no name), PCB at KERNEL (Ø35436)
```

NOTE

You cannot set breakpoints in the Debugger Service Module (DSM) or at label $TRAP in the $TRAP kernel routine. Setting breakpoints at these addresses will generate either a debugger error message, if symbols are loaded, or unpredictable results, if symbols are not loaded.

3.21 SET ODT

The SET ODT command forces the application program to halt and causes all further entries to be interpreted as commands to the target computer's Micro-ODT. PASDBG prints all target responses on the terminal. Video is reversed to signify that the ODT sequence is from the target, not the host system. Type CTRL/C or CTRL/A to exit from Micro-ODT mode.

The SET ODT command has no arguments.

Syntax

```
SET ODT
```

NOTE

Remember that SET ODT halts the target. If you want to continue target execution, you must type a P command (Micro-ODT proceed) before you exit Micro-ODT with CTRL/C or CTRL/A.

Example

```
PASDBG>set odt<RET>
@P (CRTL/C typed here, echoed below)        Displayed in
^C                                          reverse video

PASDBG>show target<RET>

 Target stopped at physical (00041456), virtual (041456) : MOV
 R0,-(SP)
 In statement 1 + 0 in
 Program PROCS2, Module PROCS2, Scope: P1
 Process 14, (no name), PCB at KERNEL (022416)
 Not using memory management hardware
```

SET PHYSICAL

3.22 SET PHYSICAL

The SET PHYSICAL command sets the current mapping to physical addressing. Program and scope settings are canceled and symbolic access is disabled. Subsequent commands must use physical addresses in place of symbol references. The SET PHYSICAL command has no arguments.

Syntax

```
     SET PHYSICAL
```

Example

```
PASDBG>set prog procs2<RET>
;PASDBG-I-NOMOD, Module and Scope set to "PROCS2"

PASDBG>ex k<RET>    !access k symbolically
(Ø43722) : Ø

PASDBG>set physical<RET>

PASDBG>show scope<RET>
?PASDBG-E-NOSCOPE, No scope set

PASDBG>ex @%43722<RET>     !access k by physical address
(Ø43722) :  Ø
```

3.23 SET PROCESS

The SET PROCESS command selects a process in the application program and sets mapping to that process. SET PROCESS allows you to access variables local to that process and any other information found on the process stack.

Note that when you set process, you do not specify the lexical name for the process. Instead, you must specify one of the identifiers associated with the process when it is created. (MicroPower/Pascal application programs can call multiple copies of a process.) These identifiers are Pascal descriptor variables, the name given the process, serial numbers, or process control block addresses.

If you do not know the valid process identifiers, use the SHOW PROCESS/ALL command to display all active processes on the target. You can then find the process that you want to set process to.

Use the SHOW PROCESS command with no arguments to display the currently set process. Note that whenever the debugger stops the target system, process is reset to the currently running process, if any. Thus, until you issue a SET PROCESS command, the currently set process is the currently running process.

Note that the SET PROGRAM command implicitly sets process to the program's static process.

You cannot use the SET PROCESS command on a process that has not yet been created.

If you SET PROCESS to use SET WATCH on a local variable, be sure to set a breakpoint on the termination point of the process and to cancel the watchpoint before the process terminates. Otherwise, unpredictable results can occur.

Syntax

```
SET PROCESS process-id
```

Command Parameter

process-id
Selects a process. Process-ids may be one of the following:

- process-descriptor -- a Pascal process-descriptor variable name.
- 'process-name' -- a runtime process name, enclosed in quotes.
- serial-number -- a runtime process serial number (unsigned integer).
- @[radix]PCB-address -- a process control block (PCB) address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal).

Example

```
PASDBG>show proc/all<RET>    !list all processes
 Process 14, (no name), PCB at KERNEL (Ø22416)
 Process 3, name = 'PROCS2', PCB at KERNEL (Ø21626)

PASDBG>show proc<RET>    !list currently set process
 Process 3, name = 'PROCS2', PCB at KERNEL (Ø21626)

PASDBG>set proc 14<RET>    !now we can access 14's local vars
```

3.24 SET PROGRAM

The SET PROGRAM command sets lexical scope to a specified program and sets mapping (PROCESS) to the specified program's static process. Lexical scope must be set -- with SET PROGRAM and SET SCOPE -- so that the debugger can differentiate between multiple occurrences of a symbol name within an application. The debugger requires that you set lexical scope before accessing any Pascal or KERNEL symbols. If you do not issue a SET PROGRAM command, you can access only physical locations with PASDBG.

The MODULE option on SET PROGRAM sets lexical scope to a particular module within the program. The default for \MODULE is the program (the module that contains the program declaration).

The KERNEL qualifier on SET PROGRAM sets the program to the MicroPower/Pascal kernel and, in a mapped system, sets KERNEL mapping. In this mode, you can access kernel symbols and addresses.

SET PROGRAM cancels the scope set with SET SCOPE and resets scope to the specified module or -- if no module is specified -- to the main block of the program.

Syntax

```
              ( program-name[\module-name] )
SET PROGRAM  <                              >
              ( KERNEL                      )
```

Command Parameters

program-name
  The name of a Pascal program in your application.

module-name
  The name of a Pascal module in that program.

KERNEL
  A keyword that sets scope (and in a mapped system, mapping) to the kernel.

Example

```
PASDBG>set prog procs2<RET>
;PASDBG-I-NOMOD, Module and scope set to "PROCS2"
```

3.25 SET SCOPE

The SET SCOPE command lets you establish a lexical path name that leads to a specific symbol. The SET SCOPE path name is composed of all the procedure, function, or process names that lead lexically to that symbol. SET SCOPE makes it possible to differentiate between multiple occurrences of a symbol name within a program or module. (See also the SET PROGRAM command.)

If the last procedure, function, or process in a lexical path has multiple occurrences, such as with recursion, you can differentiate between occurrences by specifying an occurrence number. This number is the number of occurrences before the last occurrence of the procedure, function, or process. If you do not specify a number, PASDBG defaults to the last occurrence.

SET SCOPE does not reset process -- mapping is unaffected.

Whenever the debugger stops the target system, scope is reset to the currently running program, procedure, function, or process, if any.

Syntax

```
SET SCOPE proc-name[\proc-name...][:integer]
```

Command Parameters

proc-name
: The name of a Pascal procedure, function, or process.

integer
: An occurrence number. This parameter sets scope to a particular occurrence of the last procedure, function, or process in the specified path name -- "integer" occurrences before the last one.

Example

```
[SYSTEM (MicroPower)]PROGRAM EL (INPUT,OUTPUT);
VAR PHI : REAL;
  PROCEDURE TAU;
     PROCEDURE EPSILON;
     VAR PHI : INTEGER;

     BEGIN {EPSILON}
     PHI := 1;
     END;{EPSILON}

  BEGIN {TAU}
  ...
  END;{TAU}

BEGIN {MAIN PROGRAM}
TAU;
PHI := 2.0;
END.{MAIN}
```

In this example, two distinct variables have the name PHI. Setting scope to either EL or TAU\EPSILON distinguishes between the two variables.

If PASDBG does not find the variable specified in the current scope, it removes the last procedure, function, or process from the scope list and then tries again. For example, assume a scope of:

```
PASDBG>set scope MARY\BAKER\EDDY<RET>
```

Also assume a user-specified variable name of:

```
PASDBG>EXAMINE FRED<RET>
```

Then PASDBG would attempt to find FRED, using the following scopes:

```
MARY\BAKER\EDDY
MARY\BAKER
MARY
FRED (as a global variable)
```

If PASDBG still cannot find the variable, it issues the following error message:

```
?PASDBG-E-SYMNDF, Symbol not defined in current scope
```

3.26 SET STEP

The SET STEP command establishes the default parameters for the STEP command. The parameters select the increments by which PASDBG steps and determine whether or not STEP skips over called routines.

You can set step parameters only after the application program has executed its first instruction.

You can set step parameters only for the currently executing process (SET PROCESS has no effect).

Syntax

```
              [ { INSTRUCTION } ]  [ { INTO } ]
SET STEP      [ {             } ]  [ {      } ]
              [ { STATEMENT   } ]  [ { OVER } ]
```

Command Parameters

INSTRUCTION
: Causes the execution of the program to stop after each PDP-11 instruction.

STATEMENT
: Causes the execution of the program to stop after each Pascal statement. STATEMENT is the default increment.

INTO
: Steps the program through all subroutines, including OTS routines. If a subroutine call is executed while you are stepping through a routine, PASDBG will stop and report on the first instruction of the called subroutine. INTO is the default into/over setting.

OVER
: Causes the program to step over all subroutine calls. If a subroutine call is executed while you are stepping through a routine, PASDBG will stop and report only after the subroutine returns, when the next instruction or statement in the calling routine is about to execute.

Example

```
PASDBG>show step<RET>
 Step parameters: into statement

PASDBG>set step over<RET>

PASDBG>show step<RET>
 Step parameters: over statement
```

3.27 SET TRACE

The SET TRACE command establishes a tracepoint, which is functionally equivalent to a breakpoint immediately followed by a GO. You can SET TRACE on physical/virtual locations, on a Pascal statement number, or on a Pascal label. When the instruction at the specified address is about to execute, the tracepoint triggers, and PASDBG reports the tracepoint number and any optional message you specify. Program execution then continues.

If you SET TRACE on a label and do not specify a message, PASDBG sets the message to "LABEL n procedure name", where n is the label number. If you SET TRACE on a statement and do not specify a message, PASDBG sets the message to "n procedure name", where n is the statement number.

You cannot set more than eight breakpoints and tracepoints.

Syntax

```
    SET TRACE[/qualifier...] address-expression ['message']
```

Command Parameters

address-expression
An expression giving the location at which the tracepoint is to be set. Address-expressions may be one of the following:

- statement-number -- a Pascal statement number (unsigned integer) valid within the current scope.
- LABEL label-number -- a Pascal label number (unsigned integer) defined within the current scope, preceded by the keyword "LABEL" and a space.
- MACRO-global-name -- the name of a MACRO-11 global symbol defined in the current program. Dots (".") in the symbol name must be entered as underscores ("_"); for example, PC.LNK is entered as PC_LNK.
- @[radix]address -- a physical, kernel, or virtual address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal). The address is interpreted as a physical address if physical mapping is set, a kernel address if KERNEL mapping is set, or a virtual address in the currently set process if process mapping is set.

message
A message to be displayed when the tracepoint is triggered.

Command Qualifiers

/AFTER:integer

Sets a counter on the tracepoint, causing the tracepoint to trigger only after the application program has executed the instruction the number of times specified by the integer.

/PROCESS:process-id

Causes the tracepoint to trigger only if the selected process executes the instruction. The specified process must currently exist. Process-id, which selects the process, may be one of the following:

- process-descriptor -- a Pascal process-descriptor variable name.
- 'process-name' -- a runtime process name, enclosed in quotes.
- serial-number -- a runtime process serial number (unsigned integer).
- @[radix]PCB-address -- a process control block (PCB) address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal).

Example

```
PASDBG>set trace 3 'I got here'<RET>

PASDBG>go<RET>
 [Target execution resumed - type <CR> to stop target]
 ** TRACEPOINT #0 'I got here'
```

NOTE

You cannot set tracepoints in the Debugger Service Module (DSM) or at label $TRAP in the $TRAP kernel routine. Setting tracepoints at these addresses will generate either a debugger error message, if symbols are loaded, or unpredictable results, if symbols are not loaded.

3.28 SET WATCH

The SET WATCH command establishes a watchpoint at a location you specify. A watchpoint operates as a breakpoint if the contents of a location or a Pascal variable change. The debugger stops the program and displays both the previous and current contents of the location.

You can assign a message to the watchpoint. If you do, PASDBG displays the message when the watchpoint is triggered.

You cannot use the SET WATCH command on a Pascal structure such as an array. However, you can set watch on a simple variable within the structure, such as one of the array elements.

NOTE

> You cannot watch labels or statement numbers.

Syntax

```
SET WATCH[/AFTER:integer] address-expression ['message']
```

Command Parameters

address-expression
: An expression giving the location at which the watchpoint is to be set. Address-expressions may be one of the following:

- variable-name -- the name of a Pascal variable defined within the current scope. Structure names are prohibited. However, you can specify an array element or a record field if it is a simple type.
- MACRO-global-name -- the name of a MACRO-11 global symbol defined in the current program. Dots (".") in the symbol name must be entered as underscores ("_"); for example, PC.LNK is entered as PC_LNK.
- @[radix]address -- a physical, kernel, or virtual address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal). The address is interpreted as a physical address if physical mapping is set, a kernel address if KERNEL mapping is set, or a virtual address in the currently set process if process mapping is set.

message
: A message to be displayed when the watchpoint is triggered.

Command Qualifier

/AFTER:integer
: Causes the watch to trigger only after the variable has changed value the specified number of times.

Example

```
PASDBG>set watch k<RET>

PASDBG>go<RET>
 [Target execution resumed - type <CR> to stop target]
 ** WATCHPOINT #Ø  'K'
   Old contents: Ø
   New contents: 2

 Target stopped at physical (ØØØ36744), virtual (36744) : MOV
 #144,(RØ)
 In statement 3 + Ø in
 Program PROCS2, Module PROCS2, Scope: PROCS2
 Process 3, name = 'PROCS2', PCB at KERNEL (Ø21626)
```

3.29 SHOW BREAK

The SHOW BREAK command lists all currently set breakpoints, along with the following information for each breakpoint:

- The physical address of the breakpoint
- The value set with /AFTER in SET BREAK (if not set, AFTER = 1)
- The current value of the /AFTER count
- The serial number of the process specified with /PROCESS in SET BREAK
- The message field, if any

The SHOW BREAK command has no arguments.

Syntax

```
SHOW BREAK
```

Example

```
PASDBG>show break<RET>
     #      Physaddr  AFTER   count   process S.N.  TAG
     Ø      ØØØ53Ø7Ø     1       Ø      (any)       'LABEL 1'
     1      ØØØ53Ø12     5       3      (any)       '1 PROC1'
     2      ØØØ5272Ø     1       Ø      (any)       '1 PROC2'
```

Note that breakpoints #1 and #2 are set at the same statement number but in two different procedures.

3.30 SHOW CALLS

The SHOW CALLS command causes PASDBG to display a list of the chain of routines that invoked the currently running routine. Note that this chain is not the lexical scope of the procedures. The command has no arguments.

Related commands are SHOW SCOPE, which displays the current lexical scope; SHOW TARGET, which displays the current state of the application; SHOW RUN QUEUE, which displays the currently running process; SHOW PROCESS, which displays the currently set process; SHOW PROCESS/ALL, which displays all processes in the application; SHOW PCB, which displays process control block information; and SHOW STRUCTURE, which displays the structure of a Pascal program, process, or routine.

Syntax

```
SHOW CALLS
```

Example

```
PASDBG>show calls<RET>
 in
 Process 18, (no name), PCB at KERNEL (022556)
 created by
 Process 3, name = 'GEN1  ', PCB at KERNEL (021626)

 In STORE_MESSAGE
  Called from P1
```

In this example, the currently running routine, STORE_MESSAGE, was called from the process P1, the current activation of which was created by the process 'GEN1'.

3.31 SHOW EXCEPTION

The SHOW EXCEPTION command displays the last exception message generated by the target and sent to the host system. The SHOW EXCEPTION command has no arguments.

Use the SHOW EXCEPTION command before resuming target execution with the GO command; SHOW EXCEPTION will have no effect if GO has been issued since the last exception message was received.

Syntax

```
SHOW EXCEPTION
```

Example

```
PASDBG>go<RET>
 [Target execution resumed - type <CR> to stop target]

RESOURCE Exception, Insufficient space for stack (ES$NMS)
   physical (00036674), virtual (036674) :   in statement 3 in
 Program GEN1, Module GEN1, Scope: SETUP
 Exception reported from near
   physical (00043434), virtual (043434) : in non-Pascal code .
 Process 3, name = 'GEN1  ', PCB at KERNEL (021626)
     .
     .
     .
PASDBG>show exc<RET>    !retrieve off-screen message

RESOURCE Exception, Insufficient space for stack (ES$NMS)
   physical (00036674), virtual (036674) :   in statement 3 in
 Program GEN1, Module GEN1, Scope: SETUP
 Exception reported from near
   physical (00043434), virtual (043434) : in non-Pascal code .
 Process 3, name = 'GEN1  ', PCB at KERNEL (021626)
```

SHOW EXCEPTION GROUPS

3.32 SHOW EXCEPTION GROUPS

The SHOW EXCEPTION GROUPS command displays a list of the process groups with established exception-handling processes. It also lists the types of exceptions that are handled for each process group and the packet queue that is signaled when one of the listed exceptions occurs in one of the listed process groups. To determine which processes handle the exceptions, examine the packet queues listed by SHOW EXCEPTION GROUPS with the SHOW PACKET QUEUE command.

To determine what exception-handling procedure, if any, has been established for an individual process, use the SHOW PCB command.

Syntax

```
SHOW EXCEPTION GROUPS
```

Example

```
PASDBG>sh exc group<RET>
 Exception class TRAP:
 Group 12 Packet Queue :
 Serial #13,  name = 'ex%%%%' Packet Queue  is at KERNEL (1Ø3522)
 Group 1ØØ Packet Queue :
 Serial #13,  name = 'ex%%%%' Packet Queue  is at KERNEL (1Ø3522)
 Exception class SYSTEM_SERVICE:
 Group 12 Packet Queue :
 Serial #13,  name = 'ex%%%%' Packet Queue  is at KERNEL (1Ø3522)
 Group 1ØØ Packet Queue :
 Serial #13,  name = 'ex%%%%' Packet Queue  is at KERNEL (1Ø3522)

PASDBG>sh packet queue @%1Ø3522<RET>
 Serial #13,  name = 'ex%%%%' Packet Queue  is at KERNEL (1Ø3522)
 FIFO  ordered Packet Queue has value of      Ø
 Processes Blocked on this Queue:
 Process 17, name = 'eaxch', PCB at KERNEL (1Ø4246)
 FIFO  ordered Packets waiting on this Queue:
 none.
```

In this example, exception handling has been established for process groups 12 and 1ØØ. TRAP exceptions are handled for process groups 12 and 1ØØ, as are SYSTEM_SERVICE exceptions. In this case, all TRAP and SYSTEM_SERVICE exceptions for process groups 12 and 1ØØ are handled through one packet queue, 'ex%%%%', and one handler process, 'eaxch'. In a different case, multiple packet queues and handlers might have been established.

3.33 SHOW FREE PACKETS

The SHOW FREE PACKETS command displays the number of packets that are left in the kernel packet pool. You can use this command to look at the pool and determine if you have allocated enough packets in the configuration file. The SHOW FREE PACKETS command has no arguments.

Free packet information is also displayed by the SHOW FREE STRUCTURES command.

Syntax

```
    SHOW FREE PACKETS
```

Example

```
    PASDBG>show free packets<RET>
     There are 20 free packets
```

3.34 SHOW FREE STRUCTURES

The SHOW FREE STRUCTURES command displays the amount of memory left in the kernel free-memory pool (the memory used to hold kernel structures). This memory pool can become fragmented if many structures are created and destroyed. You can use the command to look at the pool and then determine if you have allocated enough memory for the structures in the configuration file or if you have destroyed all the structures that you have finished using. SHOW FREE STRUCTURES also displays the number of packets left in the kernel packet pool. The SHOW FREE STRUCTURES command has no arguments.

Syntax

```
SHOW FREE [STRUCTURES]
```

Example

```
PASDBG>show free<RET>
 There are 2Ø free packets

 Free memory for KERNEL structures:
  Location (21374) Size 6Ø
  Location (2172Ø) Size 2768

 Total number 2, total_memory 2828
```

3.35 SHOW INACTIVE QUEUE

The SHOW INACTIVE QUEUE command lists all processes on the inactive queue and their process names, numbers, and process control block (PCB) addresses. The inactive queue contains processes that have aborted with fatal exceptions (that is, exceptions for which no exception handling was established).

The SHOW INACTIVE QUEUE command has no arguments.

Syntax

```
SHOW INACTIVE [QUEUE]
```

Example

```
PASDBG>show inactive<RET>
Processes on the inactive queue:
 Process 3, name = 'PROCS2' PCB at KERNEL (Ø36736)
 Process 9, (no name), PCB at KERNEL (Ø35116)
```

NOTE

When a process is removed from the run queue and is placed on the inactive queue, it is also deleted from the system name table and the active process list. For this reason, the commands SHOW PROCESS/ALL and SHOW NAMES, described elsewhere in this chapter, will not list the aborted process, and a SET PROCESS to the name of the aborted process will not work. To use the SET PROCESS or SHOW PROCESS commands with a process that is on the inactive queue, you must refer to the process by its process control block (PCB) address.

SHOW NAMES

3.36 SHOW NAMES

The SHOW NAMES command lists all the named kernel structures, their serial numbers, and their kernel addresses. After giving the SHOW NAMES command, you can use commands such as SHOW PCB, SHOW SEMAPHORE, SHOW PACKET QUEUE, and SHOW RING BUFFER to display more detailed information about named kernel structures. The SHOW NAMES command has no arguments.

Syntax

```
     SHOW NAMES
```

Example

```
     PASDBG>show names<RET>
      Serial #1Ø, name = '$XLCTL' PCB is at KERNEL (Ø22122)
      Serial #12, name = 'XLOØ  ' Ring Buffer is at KERNEL (Ø22332)
      Serial #2, name = '$XLADR' PCB is at KERNEL (Ø21512)
      Serial #11, name = 'XLIO  ' Ring Buffer is at KERNEL (Ø22236)
      Serial #3, name = 'GEN1  ' PCB is at KERNEL (Ø21626)
      Serial #8, name = '$XLA  ' Packet Queue is at KERNEL (Ø22Ø46)
```

3.37 SHOW PACKET QUEUE

The SHOW PACKET QUEUE command displays information about a packet queue. Use the command to display the packet queue name, address, serial number, and value; a list of all processes waiting on the packet queue; and a list of the packets waiting on the queue.

To display all currently valid packet queue names, use the SHOW NAMES command.

Syntax

SHOW PACKET QUEUE structure-id

Command Parameter

structure-id
Selects the packet queue to be displayed. Structure-ids may be one of the following:

- queue-semaphore-descriptor -- a Pascal queue semaphore descriptor variable name.
- 'queue-semaphore-name' -- a runtime queue semaphore name, enclosed in quotes.
- @[radix]queue-semaphore-address -- the kernel address of a queue semaphore, preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal).

Example

```
PASDBG>show packet queue '$XLA'<RET>

 Serial #6, name = '$XLA   ', Packet Queue is at KERNEL (Ø21456)
 Priority ordered Packet Queue has value of Ø
 Processes blocked on this Queue:
 Process 2, name = '$XLADR', PCB at KERNEL (Ø21512)
 Priority ordered packets waiting on this Queue:
 none.
```

SHOW PCB

3.38 SHOW PCB

The SHOW PCB command displays information about the process control block for the process you specified. Depending on the argument, SHOW PCB prints information about the currently set process or a specified process.

SHOW PCB lists the process's priority, state, status, mapping type (general, device, driver, or privileged), exception mask, blocking semaphore or ring buffer, suspend count, context switch options, context switch location, stack pointer, and program counter.

You can display process control block information for a process other than the currently set process by specifying a process identifier -- a Pascal descriptor variable, name variable, serial number, or control block address. Use SHOW PROCESS/ALL to list currently valid process identifiers.

Syntax

```
SHOW PCB [process-id]
```

Command Parameters

(no arguments)
Indicates the currently set process.

process-id
Selects a process. Process-ids may be one of the following:

- process-descriptor -- a Pascal process-descriptor variable name.
- 'process-name' -- a runtime process name, enclosed in quotes.
- serial-number -- a runtime process serial number (unsigned integer).
- @[radix]PCB-address -- a process control block (PCB) address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal).

Examples

```
PASDBG>show proc/all<RET>    !list current process-ids
 Process 14, (no name), PCB at KERNEL (022416)
 Process 12, (no name), PCB at KERNEL (022272)
 Process 8, name = '$XLCTL', PCB at KERNEL (021762)
 Process 3, name = 'PROCS2', PCB at KERNEL (021626)
 Process 2, name = '$XLADR', PCB at KERNEL (021512)
```

```
PASDBG>show pcb 14<RET>
 Process 14, (no name), PCB at KERNEL (Ø22416)
 Priority = 1, Status = Ready/Active, Type = General
 PC = (Ø41574), Stack pointer = (Ø45464), PSW = (ØØØ), Suspend
 count = Ø
 Termination address = (Ø4Ø26Ø), Exception group = 1
 Exceptions accepted: none
 Stack limits (Ø45514) to (Ø44676)
 Context switch options = switch location (Ø477Ø6)

PASDBG>show pcb 12<RET>
 Process 12, (no name), PCB at KERNEL (Ø22272)
 Priority = 174, Status = Wait/Active, Type = Driver
 Process is blocked on a Ring Buffer, Address = (Ø22172)
 Name ='XLOØ  '
 PC = (Ø34Ø14), Stack pointer = (Ø36ØØ6), PSW = (ØØØ), Suspend
 count = Ø
 Termination address = (Ø34Ø6Ø), Exception group = 1
 Exceptions accepted: none
 Stack limits (Ø36Ø3Ø) to (Ø35716)
 Context Switch options = none
```

SHOW PROCESS

3.39 SHOW PROCESS

The SHOW PROCESS command lists the name, serial number, and kernel PCB address for a specified process. Depending on the arguments, PASDBG provides this information on the currently set process -- which may or may not be the currently running process -- the requested process, or all processes.

To display a particular process, you do not specify the lexical name for the process. Instead, you must specify one of the identifiers associated with the process when it is created (MicroPower/Pascal application programs can create multiple copies of a process). These identifiers are Pascal descriptor variables, name variables, serial numbers, or control block addresses.

If you do not know the valid process identifiers, use the SHOW PROCESS/ALL command to display those currently valid.

Whenever PASDBG stops the target system, process is reset to the currently running process, if any. Thus, until you issue a SET PROGRAM or SET PROCESS, the currently set process is the currently running process. However, the SHOW PROCESS command should not be used as a substitute for SHOW RUN QUEUE, which always displays the running process, regardless of current process setting.

Syntax

```
     SHOW PROCESS[/ALL] [process-id]
```

Command Parameters

(no arguments or qualifiers)
    Requests information on the currently set process.

process-id
    Specifies a process. Process-ids may be one of the following:

- process-descriptor -- a Pascal process-descriptor variable name.
- 'process-name' -- a runtime process name, enclosed in quotes.
- serial-number -- a runtime process serial number (unsigned integer).
- @[radix]PCB-address -- a process control block (PCB) address preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal).

Command Qualifier

/ALL
    Directs PASDBG to display all processes.

Example

```
PASDBG>show process/all<RET>
 Process 3, Name = 'PROCS2',PCB at KERNEL (Ø36272)
 Process 9, (no name), PCB at KERNEL (Ø35116)

PASDBG>show process<RET>    !currently set process
 Process 9, (no name), PCB at KERNEL (Ø35116)

PASDBG>show run<RET>    !currently running process
 Process 3, name = 'PROCS2', PCB at KERNEL (Ø36272)
```

3.4Ø SHOW READY/ACTIVE QUEUE

The SHOW READY/ACTIVE QUEUE command lists all processes on the ready/active queue and their process names, numbers, and process control block (PCB) addresses. The SHOW READY/ACTIVE QUEUE command has no arguments.

Syntax

```
SHOW READY/ACTIVE [QUEUE]
```

Example

```
PASDBG>show ready/active queue<RET>
Processes on the Ready/Active queue:
 Process 3, name = 'PROCS2', PCB at KERNEL (Ø36736)
 Process 9, (no name), PCB at KERNEL (Ø35116)
```

3.41 SHOW READY/SUSPENDED QUEUE

The SHOW READY/SUSPENDED QUEUE command lists all processes on the ready/suspended queue and their process names, numbers, and process control block (PCB) addresses.

Syntax

```
    SHOW READY/SUSPENDED [QUEUE]
```

Example

```
    PASDBG>show ready/suspended queue<RET>
    Processes on the Ready/Suspended queue:
     Process 3, name = 'PROCS2', PCB at KERNEL (036736)
     Process 9, (no name), PCB at KERNEL (035116)
```

3.42 SHOW RING BUFFER

The SHOW RING BUFFER command displays information about a ring buffer. The command displays the ring buffer name, address, serial number, filled bytes used, low limit, high limit, the processes waiting for room on the ring buffer, the processes waiting to get an element from the ring buffer, the get and put pointers and queues, and the size of the buffer.

To display all currently valid ring buffer names, use the SHOW NAMES command.

Syntax

```
SHOW RING [BUFFER] structure-ID
```

Command Parameter

structure-ID
Selects the ring buffer to be displayed. Structure-ids may be one of the following:

- ring-buffer-descriptor -- a Pascal ring buffer descriptor variable name.
- 'ring-buffer-name' -- a runtime ring buffer name, enclosed in quotes.
- @[radix]ring-buffer-address -- the kernel address of the ring buffer, preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal).

Example

```
PASDBG>show ring rb<RET>
Serial #12, name = 'rbl    ' Ring Buffer is at KERNEL (022332)
Ring Buffer has 0 bytes used out of 8
Buffer addresses KERNEL (022366) to (022376)
Location of next get = (022366), next put = (022366)
Process currently doing get:
Process 14, (no name), PCB at KERNEL (022432)
FIFO ordered Putting Processes waiting :
none.
FIFO ordered Getting Processes waiting :
none.
```

3.43 SHOW RUN QUEUE

The SHOW RUN QUEUE command displays the currently running process's number, name, and process control block (PCB) address. The SHOW RUN QUEUE command has no arguments.

Syntax

```
SHOW RUN [QUEUE]
```

Example

```
PASDBG>show run queue<RET>
 Process 3, name = 'PROCS2', PCB at KERNEL (Ø36272)
```

3.44 SHOW SCOPE

The SHOW SCOPE command displays the current lexical scope setting. The debugger resets scope to the currently running program, process, or routine, if any, whenever it interrupts target execution. However, if the application has not started executing or if you have issued a SET PROGRAM or SET SCOPE command since execution was interrupted, SHOW SCOPE will display the lexical scope you have set. If scope is not set, PASDBG displays the message "?PASDBG-E-NOSCOPE, No scope is set".

The SHOW SCOPE command has no arguments.

Syntax

```
SHOW SCOPE
```

Example

```
PASDBG>show scope<RET>

 Program DEMO, Module DEMO, Scope: STORE_MESSAGE
```

3.45 SHOW SEMAPHORE

The SHOW SEMAPHORE command displays information about a binary or counting semaphore. The command displays the semaphore name, address, serial number, and type; the status of the semaphore variable; and a list of all processes waiting on the semaphore.

To display all currently valid semaphore names, use the SHOW NAMES command.

Syntax

```
     SHOW SEMAPHORE structure-ID
```

Command Parameter

structure-ID
: Selects the semaphore to be displayed. Structure-ids may be one of the following:

- semaphore-descriptor -- a Pascal queue semaphore descriptor variable name.
- 'semaphore-name' -- a runtime queue semaphore name, enclosed in quotes.
- @[radix]semaphore-address -- the kernel address of the semaphore, preceded by an at sign and an optional radix indicator. The four radix indicators are % (octal), %O (octal), %X (hexadecimal), and %D (decimal).

Examples

```
PASDBG> show semaphore 'flag'<RET>
 Serial #10, name = 'flag  ', Binary Semaphore is at KERNEL
 (037006)
 FIFO ordered Binary Semaphore is closed.
 Processes blocked on this Queue:
 Process 15, name = 'scan  ', PCB at KERNEL (036052)

PASDBG>show semaphore 'bonzo'<RET>
 Serial #11, name = 'bonzo ', Counting Semaphore is at KERNEL
 (036756)
 FIFO ordered Counting Semaphore has value of 0
 Processes blocked on this Queue:
 Process 14, name = 'owha  ', PCB at KERNEL (036236)
 Process 13, name = 'tagoo ', PCB at KERNEL (036422)
 Process 17, name = 'siam  ', PCB at KERNEL (036711)
```

NOTE

MicroPower/Pascal structure names are case sensitive. "Flag" is a different name from "flag."

SHOW STEP

3.46 SHOW STEP

The SHOW STEP command shows the current parameters for STEP. These parameters can be set with a SET STEP command or reset to the system default with a CANCEL STEP or an INIT command. (PASDBG performs an implicit INIT at start-up time and when it executes a LOAD command.) See the SET STEP command for more information on the parameters. The SHOW STEP command has no arguments.

Syntax

```
SHOW STEP
```

Example

```
PASDBG>show step<RET>
 Step parameters:  instruction over
```

3.47 SHOW STRUCTURE

The SHOW STRUCTURE command displays the Pascal program structure of the variable you specify. If the variable has a subordinate structure, such as with a record or an array, PASDBG displays those structures.

If you specify a process, a procedure, or a function, PASDBG displays a list, starting with the specified element, of all processes, procedures, functions, and variables lexically subordinate to the element.

Syntax

```
SHOW STRUCTURE structure-name
```

Command Parameter

structure-name
Selects the structure to be displayed. Structure names can be any of the following:

- variable-name -- the name of a Pascal variable
- routine-name -- the name of a Pascal procedure or function
- process-name -- the name of a Pascal process
- program-name -- the name of a Pascal program

Examples

```
PASDBG>SHOW STR FOO<RET>
```

In this example, FOO is a record containing an integer field, a real field, and a Boolean field. PASDBG displays:

```
FOO : RECORD OF
  BAR3 : BOOLEAN
  BAR2 : REAL
  BAR1 : INTEGER
```

Shown below is the structure of a program named GEN1.

```
PASDBG>show str gen1<RET>
  MAIN BLOCK GEN1
    INDEX : SUBRANGE Ø..65535
    MESSAGES : ARRAY [1..11] OF
       : ARRAY [1..16] OF
         : CHAR
    I : SUBRANGE Ø..65535
    PROCEDURE STORE_MESSAGE
      MSG : ARRAY [1..16] of
         : CHAR
    PROCESS GEN2
      NAME : ARRAY [1..6] OF
         : CHAR
  PRIORITY : INTEGER
  STACK_SIZE : INTEGER
  DESC : INTEGER
  I : SUBRANGE Ø..65535
PROCEDURE SETUP
PROCEDURE PRINT_MESSAGES
  I : SUBRANGE Ø..65535
```

3.48 SHOW TARGET

The SHOW TARGET command gives information on the state of the hardware and software active on the application system. SHOW TARGET indicates the location at which the target is currently stopped, the assembler instruction at that location, the process name and serial number, and whether or not memory management hardware is in use.

The SHOW TARGET command has no arguments.

Syntax

```
     SHOW TARGET
```

Example

```
     PASDBG>show target<RET>

      Target stopped at physical (00036562), virtual (036562) : CLR
      4(SP)
      In statement 1 + 0 in
      Program PROCS1, Module PROCS1, Scope: P1
      Process 14, (no name), PCB at KERNEL (022416)
      Not using memory management hardware
```

SHOW TRACE

3.49 SHOW TRACE

The SHOW TRACE command lists all currently set tracepoints, along with the following information for each tracepoint:

- The physical address of the tracepoint
- The value set with /AFTER in SET TRACE (if not set, AFTER = 1)
- The current value of the /AFTER count
- The serial number of the process specified with /PROCESS in SET TRACE
- The message field, if any

The SHOW TRACE command has no arguments.

Syntax

```
SHOW TRACE
```

Example

```
PASDBG>show trace<RET>
  #      Physaddr  AFTER    count   process S.N.  TAG
  Ø      ØØØ53Ø7Ø      1        Ø     (any)       'LABEL 1'
  1      ØØØ53Ø12      5        3     (any)       '1 PROC1'
  2      ØØØ5272Ø      1        Ø     (any)       'Marker'
```

3.5Ø SHOW WATCH

The SHOW WATCH command lists the following information about all currently set watchpoints:

- The physical address of the watchpoint
- The value set with /AFTER in SET WATCH (if not set, AFTER = 1)
- The current value of the /AFTER count
- The message field, if any

The SHOW WATCH command has no arguments.

Syntax

```
SHOW WATCH
```

Example

```
PASDBG>show watch<RET>
  #      Physaddr  AFTER    count   process S.N.   TAG
  Ø      ØØØ53Ø7Ø      1        Ø       (any)      'Marker'
  1      ØØØ53Ø12      5        3       (any)      'Proc 1'
  2      ØØØ5272Ø      1        Ø       (any)      'Proc 2'
  3,4    ØØØ5276Ø      1        Ø       (any)      'a real'
```

The "process S.N." column in the example above has no meaning for watchpoints. It is significant only for breakpoint and tracepoint display (see the SHOW BREAK and SHOW TRACE commands).

Note that real numbers use two watchpoints.

SPAWN

3.b SPAWN

The SPAWN command is implemented only in the RSX and VMS versions of PASDBG. This command permits you to issue any single-line RSX or VMS command while in PASDBG and then return to the debugging session. The command allows you to use an editor, for example, from within PASDBG.

To use SPAWN under PASDBG-VMS, you must have the TMPMBX (temporary mailbox) privilege. Also, SPAWN creates a subprocess, so your subprocess quota has to be large enough to allow creation of the additional process. If your quota is not sufficient, PASDBG prints the error message "?PASDBG-E-NODCL, Unable to Spawn subprocess".

Syntax

```
     SPAWN cmd
```

Command Parameter

cmd
    A single-line RSX or VMS (DCL) command.

3.51 STEP

The STEP command allows you to execute the application program one increment at a time, as defined by SET STEP. When the application system finishes performing the step, PASDBG issues information as if a breakpoint had occurred. You can also specify stepping through multiple increments with an integer qualifier.

The /INSTRUCTION, /STATEMENT, /INTO, and /OVER qualifiers override the default parameters specified with SET STEP. The default parameters are restored after the STEP command has executed.

Use the STEP command only within a process. If you execute a STEP and another process begins execution, the program proceeds until the first process begins executing again.

The exception to this rule occurs if watchpoints are set in your application. Because both WATCH and STEP use the same trap mechanism, the watchpoints cause you to step into the new process.

If another debugger command interrupts a STEP -- a breakpoint in a called routine, for example -- the STEP command is canceled.

Syntax

```
     STEP[/qualifier[/qualifier]] [integer]
```

Command Parameter

integer
    The number of step increments to perform before stopping application execution.

Command Qualifiers

/INSTRUCTION
    Causes the program to stop after one PDP-11 instruction.

/STATEMENT
    Causes the program to stop after one Pascal statement.

/INTO
    Steps the program into any subroutine call, including an OTS routine call. If the instruction or statement about to execute calls a subroutine, PASDBG will stop and report on the first instruction of the called subroutine. If no subroutine is called, /INTO has no effect.

/OVER
    Causes the program to step over any subroutine call, stopping only when the subroutine returns and the next instruction or statement in the currently active routine is about to execute. If no subroutine is called, /OVER has no effect.

Example

```
PASDBG>show step<RET>  !currently at stmt 2+Ø in STORE_MESSAGE
 Step parameters: into statement

PASDBG>s/instr<RET>

 Target stopped at physical (ØØØ3674Ø), virtual (Ø3674Ø) : ASL R3
 In statement 2 + 4 in
 Program GEN1, Module GEN1, Scope: STORE_MESSAGE
 Process 18, (no name), PCB at KERNEL (Ø22556)

PASDBG>s<RET>   !return to statement increments

 Target stopped at physical (ØØØ36776), virtual (Ø36776) : JSR
 PC,41166
 In statement 3 + Ø in
 Program GEN1, Module GEN1, Scope: STORE_MESSAGE
 Process 18, (no name), PCB at KERNEL (Ø22556)
```

APPENDIX

TARGET INTERFACE SPECIFICATIONS

The MicroPower/Pascal debugger, PASDBG, runs on an RT-11, RSX-11M/M-PLUS, or VAX/VMS system, hereafter referred to as the host system, which is totally distinct from the system running MicroPower/Pascal and the user application programs, hereafter referred to as the target system. The two systems communicate with a high baud rate, asynchronous, serial line (RS-232). The debugger service module (DSM) -- a part of the MicroPower/Pascal executive with addressing access to the entire target system address space -- is responsible for controlling the target's part of the serial communication. The DSM, under commands from the debugger over the serial link, examines and modifies target memory, sets and reports breakpoints, tracepoints, and watchpoints, and adjusts memory mapping so that the debugger can reference virtual addresses within processes and target kernel data. (Other tasks the DSM performs are listed below.)

The communication scheme developed to handle messages between the debugger and the target DSM is an ASCII-character-oriented scheme subject to the following protocols.

A.1 PROTOCOLS

Two basic protocols control the host to target link: the inquiry/response protocol and the asynchronous message protocol. These protocols are described below.

A.1.1 Inquiry/Response Protocol

The debugger initiates the inquiry/response protocol as a request for data or a command for action by the DSM. Messages in this protocol can be passed only while the DSM is in the "idle" state, that is, after stopping for break or watchpoints or after a HALT. The HALT command itself is the exception to this rule.

Individual message formats differ according to function, but all have the same basic format. The script is as follows:

```
     Debugger:  {bom} {message} cc {eom}
     DSM:  {bom} {response} cc {eom}
```

In the example, {bom} is the "beginning of message" character, and cc is a 2-character hexadecimal checksum derived by adding the ASCII characters in the message and inverting the result. The checksum value does not include the {bom} character, itself, or the {eom} "end of message" character.

On a checksum error (host to target), the receiving end sends a "<" or a ">" (target to host), and the sender sends the message back in the same format. The messages "<" and ">" are known as "NACK" (negative acknowledgment). NACKs do not have checksums, because it is better for a sender to send again rather than to NACK a NACK.

The message receiver is also responsible for detecting the case where too much time has elapsed while waiting to receive a character. This is a "timeout," which causes the receiver to send a NACK to the sender, causing the sender to send its message again.

A.1.2 Asynchronous Message Protocol

The asynchronous message protocol, initiated by the DSM only while the target is running, is a report that some event(s) the debugger is waiting for has occurred. Examples of this type of event include the occurrence of a breakpoint, a watchpoint, and a target exception. The DSM will stop after sending an asynchronous message but can send more than one (multiple breaks at the same address, one instruction that changes more than one watchpoint, and so on). For this reason, there is a handshaking protocol while the DSM transmits the break, watch, or stop messages. This handshaking sequence is as follows:

```
DSM:    {bom} {message} cc {eom}
DBG:    "&" (or ">" to request a resend of above message)
        {bom} {message} cc {eom}
DBG:    "&" (or ">")
            ...
DSM:    {bom} H cc {eom} (indicate DSM now idle)
DBG:    "&"
```

If an asynchronous message receives no response within a timeout period, the DSM will continue to send the asynchronous message until it is acknowledged. Note that the use of the 2-digit hex inverted checksum, as well as the receiver timeout checking on both ends, is identical to the inquiry/response case. The "ACKs" ("&") are sent alone, without {bom}, checksum, or {eom} -- just like "NACKs".

NOTE

Currently, {eom} = "/" for host-to-target messages and octal 32 (ASCII CTRL/Z) for target-to-host messages; {bom} = "!".

A.2 MESSAGES

A.2.1 Inquiry/Response Messages

The following are the currently defined inquiry/response messages. Note that the {bom} character, the checksum characters, and the {eom} character have been removed.

All numeric information is passed in ASCII hexadecimal (Radix 16).

Clear Break

```
     DBG: 9n
     DSM: 1

          n = breakpoint ordinal (Ø-7)
```

Clear Watch

```
     DBG: Bn
     DSM: 1

          n = watchpoint ordinal (table entry)
```

Fetch from KERNEL

```
     DBG: Ckkkk
     DSM: 1dddd
or
          Ø   {memory access error}

          kkkk = Kernel address for fetch
          dddd = Data from fetch
```

Fetch Register (current mapped process)

```
     DBG: Et
     DSM: 1dddd

          t = register number Ø-7 (or 8=PSW) for fetch
          d = register contents from fetch
```

Fetch via Physical Address

```
     DBG: Øppppoo
     DSM: 1dddd
or
          Ø   {memory access error}

          ppppoo = physical address for fetch
          pppp = PAR value which is high 13 bits of address
          oo = 6-bit offset in block
          dddd = result of fetch
```

Fetch via Virtual Address

```
     DBG: 4vvvv
     DSM: 1dddd
or
          Ø   {memory access error}

          vvvv = virtual address for fetch
          dddd = data from fetch
```

GO

```
     DBG: 6
     DSM: 1
```

Interrogate PC

```
     DBG: 2
     DSM: 1vvvvppppookkkk


          vvvv = virtual PC at STOP
          pppp = par value of phys PC (high 16 bits)
          oo = low 6 bits of physical PC
          kkkk = KERNEL PCB address of running process ($RUN)
```

Restart (restore to "just loaded" state)

```
     DBG: K
     DSM: 1
```

Return mapping flag

```
     DBG: 7
     DSM: 1    {target using memory management hardware}
or
          Ø    {target using physical address mapping only}
```

Set Break

```
     DBG: 8nppppoo
     DSM: 1
or
          Ø    {memory access error}

          n = breakpoint ordinal (table entry) Ø-7
          pppp = par value of phys addr (high 16 bits)
          oo = low 6 bits of physical address
```

Set Mapping by Process ID

```
     DBG: 3kkkk
     DSM: 1

          kkkk = kernel address of PCB for process
```

Set Watch

```
     DBG: Anppppoo
     DSM: 1
or
          Ø    {memory access error}

          n = watchpoint ordinal (table entry)
          pppp = par value of phys addr (high 16 bits)
          oo = low 6 bits of physical address
```

Step (report every instruction in this process)

```
     DBG: I
     DSM: 1
```

Step (report on next RTS PC)

```
     DBG: J
     DSM: 1
```

STOP

    DBG: *
    DSM: {asynchronous sequence}

Note that the stop message ("*") is not sent via the normal message protocol; only a single "*" is sent.

After the debugger requests a stop, the DSM and the debugger enter the asynchronous handshaking sequence (the asynchronous protocol described above). The conclusion of the sequence -- the reception of the final "{bom} H cc {eom}" message -- indicates to the debugger that the target is now idle.

Store Register

    DBG: Ftdddd
    DSM: 1

    t = register number for store (Ø-5,8; No SP, PC)
    d = data for store

Store to KERNEL

    DBG: Dkkkkdddd
    DSM: 1
or
    Ø   {memory access error}

    kkkk = KERNEL addr for store
    dddd = data for store

Store via Physical Address

    DBG: 1ddddppppoo
    DSM: 1
or
    Ø   {memory access error}

    ppppoo = physical address for fetch.
    pppp = PAR value which is high 13 bits of address
    oo = 6-bit offset in block
    dddd = data for store

Store via Virtual Address

    DBG: 5vvvvdddd
    DSM: 1
or
    Ø   {memory access error}

    vvvv = virtual address for store
    dddd = data for store

Translate kernel to physical

    DBG: Hkkkk
    DSM: 1ppppoo

    kkkk = kernel address to be translated
    pppp = par value of phys addr (high 16 bits)
    oo = low 6 bits of physical address

Translate virtual to physical

```
     DBG: Gvvvv
     DSM: lppppoo

          vvvv = virtual address to be converted
          pppp = par value of phys addr (high 16 bits)
          oo = low 6 bits of physical address
```

A.2.2 Asynchronous Messages

The following formats are for the asynchronous messages -- those messages, coming from the target, that are not a direct response to an inquiry by the host debugger. Note that asynchronous messages will not be sent between a host inquiry and the DSM response.

Abort (exception detected in exception handler)

```
     DSM: A
     DBG: &      {no eom, cc, or bom}
```

Break

```
     DSM: Bn
     DBG: &      {no eom, cc, or bom}

          n = breakpoint ordinal
```

DSM init (power-up sequence executed via load or INIT/RESTART)

```
     DSM: In
     DBG: &      {no eom, cc, or bom}

          n = DSM version number
```

Exception

```
     DSM: E
     DBG: &      {no eom, cc, or bom}
```

Kernel abort (kernel detects fatal trap in kernel)

```
     DSM: K
     DBG: &      {no eom, cc, or bom}
```

Step (step report)

```
     DSM: S
     DBG: &      {no eom, cc, or bom}
```

Watch

```
     DSM: Wn
     DBG: &      {no eom, cc, or bom}

          n = tracepoint ordinal
```

INDEX

HOW TO ORDER ADDITIONAL DOCUMENTATION

From	Call	Write
Chicago	312–640–5612 8:15 A.M. to 5:00 P.M. CT	Digital Equipment Corporation Accessories & Supplies Center 1050 East Remington Road Schaumburg, IL 60195
San Francisco	408–734–4915 8:15 A.M. to 5:00 P.M. PT	Digital Equipment Corporation Accessories & Supplies Center 632 Caribbean Drive Sunnyvale, CA 94086
Alaska, Hawaii	603–884–6660 8:30 A.M. to 6:00 P.M. ET *or* 408–734–4915 8:15 A.M. to 5:00 P.M. PT	
New Hampshire	603–884–6660 8:30 A.M. to 6:00 P.M. ET	Digital Equipment Corporation Accessories & Supplies Center P.O. Box CS2008 Nashua, NH 03061
Rest of U.S.A., Puerto Rico*	1–800–258–1710 8:30 A.M. to 6:00 P.M. ET	

*Prepaid orders from Puerto Rico must be placed with the local DIGITAL subsidiary (call 809–754–7575)

From	Call	Write
Canada		
British Columbia	1–800–267–6146 8:00 A.M. to 5:00 P.M. ET	Digital Equipment of Canada Ltd 940 Belfast Road Ottawa, Ontario K1G 4C2 Attn: A&SG Business Manager
Ottawa–Hull	613–234–7726 8:00 A.M. to 5:00 P.M. ET	
Elsewhere	112–800–267–6146 8:00 A.M. to 5:00 P.M. ET	
Elsewhere		Digital Equipment Corporation A&SG Business Manager*

*c/o DIGITAL's local subsidiary or approved distributor

MicroPower/Pascal–RT
Debugger User's Guide
AA–M393B–TC

READER'S COMMENTS

OTE: This form is for document comments only. DIGITAL will use comments submitted on this form at the company's discretion. If you require a written reply and are eligible to receive one under Software Performance Report (SPR) service, submit your comments on an SPR form.

id you find this manual understandable, usable, and well organized? Please make suggestions for improvement.

____
____
____
____
____
____
____
____

id you find errors in this manual? If so, specify the error and the page number.

____
____
____
____
____
____
____
____
____

Please indicate the type of user/reader that you most nearly represent.

- _ Assembly language programmer
- _ Higher-level language programmer
- _ Occasional programmer (experienced)
- _ User with little programming experience
- _ Student programmer
- _ Other (please specify) ____________________

Name ____________________ Date ____________________

Organization ____________________ Telephone ____________________

Street ____

City ____________________ State __________ Zip Code __________
or Country

— — Do Not Tear — Fold Here and Tape — — — — — — — — — — — — — — — — — —

digital

No Postage
Necessary
if Mailed in the
United States

BUSINESS REPLY MAIL
FIRST CLASS PERMIT NO.33 MAYNARD MASS.

POSTAGE WILL BE PAID BY ADDRESSEE

SSG/ML PUBLICATIONS, MLO5–5/E45
DIGITAL EQUIPMENT CORPORATION
146 MAIN STREET
MAYNARD, MA 01754

— — Do Not Tear — Fold Here —