HEWLETT PACKARD

Reference

68030 Emulator

HP 64430

68030 Emulator

Reference

HEWLETT
PACKARD

HP Part No. 64430-97009
Printed In U.S.A.
June 1991

Edition 3

Certification and Warranty

Certification

Hewlett-Packard Company certifies that this product met its published specifications at the time of shipment from the factory. Hewlett-Packard further certifies that its calibration measurements are traceable to the United States National Bureau of Standards, to the extent allowed by the Bureau's calibration facility, and to the calibration facilities of other International Standards Organization members.

Warranty

This Hewlett-Packard system product is warranted against defects in materials and workmanship for a period of 90 days from date of installation. During the warranty period, HP will, at its option, either repair or replace products which prove to be defective.

Warranty service of this product will be performed at Buyer's facility at no charge within HP service travel areas. Outside HP service travel areas, warranty service will be performed at Buyer's facility only upon HP's prior agreement and Buyer shall pay HP's round trip travel expenses. In all other cases, products must be returned to a service facility designated by HP.

For products returned to HP for warranty service, Buyer shall prepay shipping charges to HP and HP shall pay shipping charges to return the product to Buyer. However, Buyer shall pay all shipping charges, duties, and taxes for products returned to HP from another country. HP warrants that its software and firmware designated by HP for use with an instrument will execute its programming instructions when properly installed on that instrument. HP does not warrant that the operation of the instrument, or software, or firmware will be uninterrupted or error free.

Limitation of Warranty

The foregoing warranty shall not apply to defects resulting from improper or inadequate maintenance by Buyer, Buyer-supplied software or interfacing, unauthorized modification or misuse, operation outside of the environment specifications for the product, or improper site preparation or maintenance.

No other warranty is expressed or implied. HP specifically disclaims the implied warranties of merchantability and fitness for a particular purpose.

Exclusive Remedies

The remedies provided herein are buyer's sole and exclusive remedies. HP shall not be liable for any direct, indirect, special, incidental, or consequential damages, whether based on contract, tort, or any other legal theory.

Product maintenance agreements and other customer assistance agreements are available for Hewlett-Packard products.

For any assistance, contact your nearest Hewlett-Packard Sales and Service Office.

Notice

Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

Hewlett-Packard assumes no responsibility for the use or reliability of its software on equipment that is not furnished by Hewlett-Packard.

Hewlett-Packard Company
Logic Systems Division
8245 North Union Boulevard
Colorado Springs, CO 80920, U.S.A.

Printing History

New editions are complete revisions of the manual. The date on the title page changes only when a new edition is published.

A software code may be printed before the date; this indicates the version level of the software product at the time the manual was issued. Many product updates and fixes do not require manual changes, and manual corrections may be done without accompanying product changes. Therefore, do not expect a one-to-one correspondence between product updates and manual revisions.

Edition 1	64430-97001, February 1990
Edition 2	64430-97006, February 1991
Edition 3	64430-97009, June 1991

Safety

Summary of Safe Procedures

The following general safety precautions must be observed during all phases of operation, service, and repair of this instrument. Failure to comply with these precautions or with specific warnings elsewhere in this manual violates safety standards of design, manufacture, and intended use of the instrument. Hewlett-Packard Company assumes no liability for the customer's failure to comply with these requirements.

Ground The Instrument

To minimize shock hazard, the instrument chassis and cabinet must be connected to an electrical ground. The instrument is equipped with a three-conductor ac power cable. The power cable must either be plugged into an approved three-contact electrical outlet. The power jack and mating plug of the power cable meet International Electrotechnical Commission (IEC) safety standards.

Do Not Operate In An Explosive Atmosphere

Do not operate the instrument in the presence of flammable gases or fumes. Operation of any electrical instrument in such an environment constitutes a definite safety hazard.

Keep Away From Live Circuits

Operating personnel must not remove instrument covers. Component replacement and internal adjustments must be made by qualified maintenance personnel. Do not replace components with the power cable connected. Under certain conditions, dangerous voltages may exist even with the power cable removed. To avoid injuries, always disconnect power and discharge circuits before touching them.

Designed to Meet Requirements of IEC Publication 348

This apparatus has been designed and tested in accordance with IEC Publication 348, safety requirements for electronic measuring apparatus, and has been supplied in a safe condition. The present instruction manual contains some information and warnings which have to be followed by the user to ensure safe operation and to retain the apparatus in safe condition.

Do Not Service Or Adjust Alone

Do not attempt internal service or adjustment unless another person, capable of rendering first aid and resuscitation, is present.

Do Not Substitute Parts Or Modify Instrument

Because of the danger of introducing additional hazards, do not install substitute parts or perform any unauthorized modification of the instrument. Return the instrument to a Hewlett-Packard Sales and Service Office for service and repair to ensure that safety features are maintained.

Dangerous Procedure Warnings

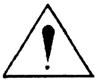
Warnings, such as the example below, precede potentially dangerous procedures throughout this manual. Instructions contained in the warnings must be followed.

Warning

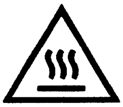
Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

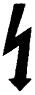
The following is a list of general definitions of safety symbols used on equipment or in manuals:



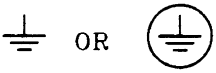
Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.



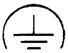
Hot Surface. This symbol means the part or surface is hot and should not be touched.



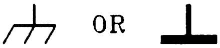
Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).



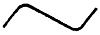
Protective conductor terminal. For protection against electrical shock in case of a fault. Used with field wiring terminals to indicate the terminal which must be connected to ground before operating the equipment.



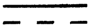
Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating the equipment.



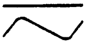
Frame or chassis terminal. A connection to the frame (chassis) of the equipment which normally includes all exposed metal structures.



Alternating current (power line).



Direct current (power line).



Alternating or direct current (power line).

Note

The Note sign denotes important information. It calls your attention to a procedure, practice, condition, or similar situation which is essential to highlight.

Caution

The Caution sign denotes a hazard. It calls your attention to an operating procedure, practice, condition, or similar situation, which, if not correctly performed or adhered to, could result in damage to or destruction of part or all of the product.

Warning

The Warning sign denotes a hazard. It calls your attention to a procedure, practice, condition or the like, which, if not correctly performed, could result in injury or death to personnel.

Using This Manual

This manual is a detailed reference for the 68030 emulator commands. The detailed syntax descriptions apply to the emulator functions only. See the *Analysis Reference Manual for 32-Bit Microprocessors* for detailed descriptions of analysis commands.

Organization

Chapter 1

"Introducing 68030 Emulation" has a brief functional and physical description of the emulation system. It also contains information on transparency and real-time emulation mode considerations.

Chapter 2

"Emulation Command Syntax" describes the emulation commands in detail with command descriptions, command syntax diagrams, and examples.

Appendix A

"User Interface/HP-UX Cross Reference" translates the HP 64000-UX system softkeys into commands that can be entered from the HP-UX prompt.

Appendix B

"Using Control Characters And Other Commands" describes how to use control characters in the emulation session. It also lists HP-UX and HP 64000-UX system commands available in an emulation session.

Understanding The Examples

This manual assumes that you are using the User-Friendly Interface Software (HP 64808S), which you start using the HP 64000-UX **pmon** command. The manual shows you how to enter HP 64000-UX system commands (edit, compile, assemble, link, msinit, msconfig, etc.) by telling you to press various softkeys.

If you are not using "pmon," you will find the User Interface/ HP-UX Cross Reference appendix especially useful. The cross reference table shows how the "pmon" softkeys translate into commands that you can enter at the HP-UX prompt.

The examples in this manual use the following structure:

```
copy display to trcfile1
```

copy display to — Softkeys appear in bold italic type in examples. Commands appear in **bold** in text. You will not be prompted to use the **---ETC---** softkey to search for the appropriate softkey template. Three softkey templates are available at the HP 64000-UX system monitor level.

trcfile1 — This is the name of a file, which you must type in. There are no softkeys for this type of selection since it is variable. However, a softkey prompt such as **<FILE>** will appear as a softkey selection.

For most commands, you must press the **Return** (or **Enter**) key before the command is executed.

Contents

1 Introducing 68030 Emulation

2 Emulation Command Syntax

Notes

Introducing 68030 Emulation

Introduction

This chapter answers the following questions:

- What is an emulation system?
- What does an emulator enable you to do?
- Does the emulator system run interactively with other HP 64000-UX Microprocessor Development Environment modules?
- Does the emulator affect your program?
- What happens while your program is running?
- What does the emulator do to your microprocessor system?
- What are the steps in using the emulator?

What Is An Emulation System?

Physical Description

The 68030 emulation system is a separate functional module within the HP 64000-UX Microprocessor Development Environment. The emulation system has several hardware modules, the emulation software, and technical manuals. A typical 68030 emulation system has the following hardware modules:

- The emulation subsystem for your microprocessor.
- Integrated analysis board.
- Integrated analysis expansion board.
- Analysis interconnect board.
- Processor specific analysis bus generator board.
- Processor active probe.

The emulation system may be used interactively with other HP 64000-UX emulation and analysis systems for more sophisticated measurements.

Functional Description

The emulator helps you develop your (target system) hardware and software design. The emulator can be used in-circuit, alone, or with other development tools to debug and integrate your target system hardware with the software program modules.

Independent Operation

The emulation and analysis functions are independent of the HP 64000-UX operating system. Once you configure and start the emulator and analyzer, they operate without interaction from the operating system. A multiprocessor system controls the emulation system and the HP 64000-UX operating system.

Emulation Probe

The emulator replaces the microprocessor in your target system with a device that acts like the microprocessor, but can be controlled by you from the development station. This is done through the emulation pod and active probe, which is part of the cable extending from the emulation pod. The active probe contains the emulation microprocessor that drives your target system. The probe is plugged into your target system microprocessor socket.

What Tasks Does The Emulator Do?

You use the emulator for software and hardware debugging and system integration. To do this, you use the emulator features:

- Program Loading and Execution. You develop programs on the HP 64000-UX system using the editor, compilers, assembler, and linker. Or, you can develop code on other systems and transfer it to the HP 64000-UX host. Then you load these programs into memory using the emulator and execute them in the emulation environment.

- Run/Stop Controls. Programs may be run from address or symbolic locations. Emulation can be stopped by breaking into the emulation monitor or by resetting the processor.

- Memory Display/Modification. You can display locations or blocks of memory and modify those locations that can be changed.

- Global and Local Symbols Display. You can display the addresses associated with your program's global and local symbols while in emulation.

- Internal Resource Display/Modification. Allows you to display internal resources of the processor, such as registers. You also can modify them if desired.

- Analysis (with optional integrated analyzer boards). Lets you observe and display real-time activity on the emulation processor bus.

- Program Stepping. Allows you to execute code instruction-by-instruction. You can view the internal machine state between instructions.

- Resource Mapping. Allows you to use emulation memory, target memory, or both by defining the characteristics of the blocks of memory.

- Memory Characterization. You can assign emulation memory as ROM or RAM. You can test "ROM" code without using ROM hardware.

- Hardware and Software Breakpoints. You can transfer program execution to an emulation monitor routine on the occurrence of a particular machine state or range of states.

- Clock Source Selection. The emulator provides an internal clock for out-of-circuit use. When your target system design is ready, you can select an external clock.

Does the Emulator Work With Other HP 64000-UX Modules?

The HP 64000-UX Microprocessor Development Environment allows you to use other emulators and analyzers to make interactive measurements. Interaction allows the integration of development work on designs, more elaborate and detailed analysis of a design, or both. You can:

- Begin multiple measurements simultaneously.
- Use the results of one measurement to control another.
- Coordinating execution of a program with the beginning of a measurement.

How Does the Emulator Affect Your Program?

The operating mode you select influences the way the emulator interacts with your program. The emulator never permanently alters your program.

Real-Time Mode Vs. Nonreal-Time Mode

The emulator operates in one of two modes: real-time or nonreal-time. Real-time refers to the continuous execution of your target system program without interference from the host. (You can use some commands to interrupt the program if needed).

Interference occurs when a break to the emulation monitor is initiated either by you or automatically. The emulation monitor is a program that enables you to access the internal processor registers and target system memory.

When the processor is running in the emulation monitor, it is not executing your program in real time. The *68030 Emulator User's Guide* describes the monitor program.

Real-Time Mode Capabilities

Commands that can be used in real-time mode are:

run, some display, some modify, specify,
execute, trace, load trace, stop_trace

Real-Time Mode Restrictions

Some commands cannot be used in real-time mode. You must first break into the emulation monitor.

Caution ✋

DAMAGE TO TARGET SYSTEM CIRCUITRY. When the emulator detects a guarded memory access or other illegal condition, or when you request a memory access that breaks the emulator into the monitor, the emulator stops executing your program and enters the monitor. Use caution if you have circuitry that can be damaged because the emulator is not executing your application code. Restrict the emulator to real-time mode, and do not break into the emulation monitor.

The features that cannot be performed in real-time mode are:

- Target memory accesses—display, copy, load, modify, and store.
- Logical emulation memory accesses with MMU enabled.
- Register accesses—display, copy, and modify.
- Software breakpoints—set and reset.

You can use these features when the emulator is configured for real time mode by causing a monitor break:

- Use the break softkey.
- Set an analysis break.
- Cause a memory break (your program accesses guarded memory or writes to ROM).
- Set and execute a software breakpoint.

What Is Happening While Your Program Is Running?

During Target Program Execution

During normal program execution, the emulation processor generates address information for each cycle. The emulator pod hardware differentiates between your target system and emulation resources based on the address. If the pod identifies a target system resource with the current address, it enables the data path buffers between your target system and the emulator processor. Otherwise, it enables the data path buffers between the emulation processor and the emulation bus.

As your program runs, the integrated analysis circuitry observes the activity on the emulation analysis bus. You can tell the analyzer to store this program flow. The information can be displayed later without interrupting the program.

During Emulation Monitor Program Control

Some emulator features are implemented by seizing control of the emulation processor from your program and transferring control to the emulation monitor. The emulation monitor program links the emulation processor to the HP 64000-UX operating system.

The emulation monitor has several separate routines. Some routines are executed automatically whenever the monitor program is entered. They extract the internal processor information that existed at entry. You can display this information to help analyze your program. For instance, if the emulator entered the monitor after the execution of a program instruction, the internal machine state that existed then would be available.

How Does The Emulator Affect Your Microprocessor System?

The emulator must look like the microprocessor that will eventually control your system, as seeŋ by your target system hardware. The function, signal quality, signal timing, loading, drive capacity, and other factors at the plug-in connector should be identical with the actual processor. This characteristic is called *transparency.*

Functional Transparency

Functional transparency is the ability of the emulator to function as your processor would when the emulator is connected to your target system. This means that the emulator must execute your program, generate outputs, and respond to inputs exactly as the actual target processor would. The emulator must simultaneously give you complete information about the clock-by-clock operation of your target system. HP 64000-UX 32-bit emulators are designed to perform their functions with minimum impact on functional transparency.

The *68030 Emulator User's Guide* discusses emulation functions that may affect your target system operation.

Timing Transparency

Timing transparency refers to the timing relationships between signals at your target system plug-in. The timing relationships of signals at the emulation probe are designed to be nearly identical to those of the microprocessor in your system.

Electrical Transparency

Electrical transparency refers to the electrical characteristics of the emulator target plug pins compared to the pins of the actual target processor. These characteristics include such things as rise and fall times, input loading, output drive capacity, and transmission line considerations. The electrical parameters at the emulation target plug pins are designed to be as close as possible to the microprocessor it replaces in your target system.

What Are The Steps To Using The Emulator?

There are three steps to the emulation process (see figure 1-1):

- Prepare the software.
- Prepare the emulator.
- Use the emulator.

Prepare the Software

Preparing the software consists of creating and entering a program, assembling or compiling the program, and linking the assembled or compiled modules. See the appropriate Assembler/Linker or Compiler Manual for more information.

Prepare the Emulator

You prepare the emulator by initializing and defining a measurement system to the HP 64000-UX operating software. See the *HP 64000-UX Measurement System Operating Manual*. After the emulator is properly defined, you configure the emulator for your application. The *68030 Emulator User's Guide* discusses emulator configuration.

Use the Emulator

To use the emulator, you load your absolute code into emulation and/or target system memory. Then, you use the emulation features to observe the program as it runs, display the contents of the registers and/or memory, and debug your hardware and software. Emulator use is covered in this manual and the *68030 Emulator User's Guide.*

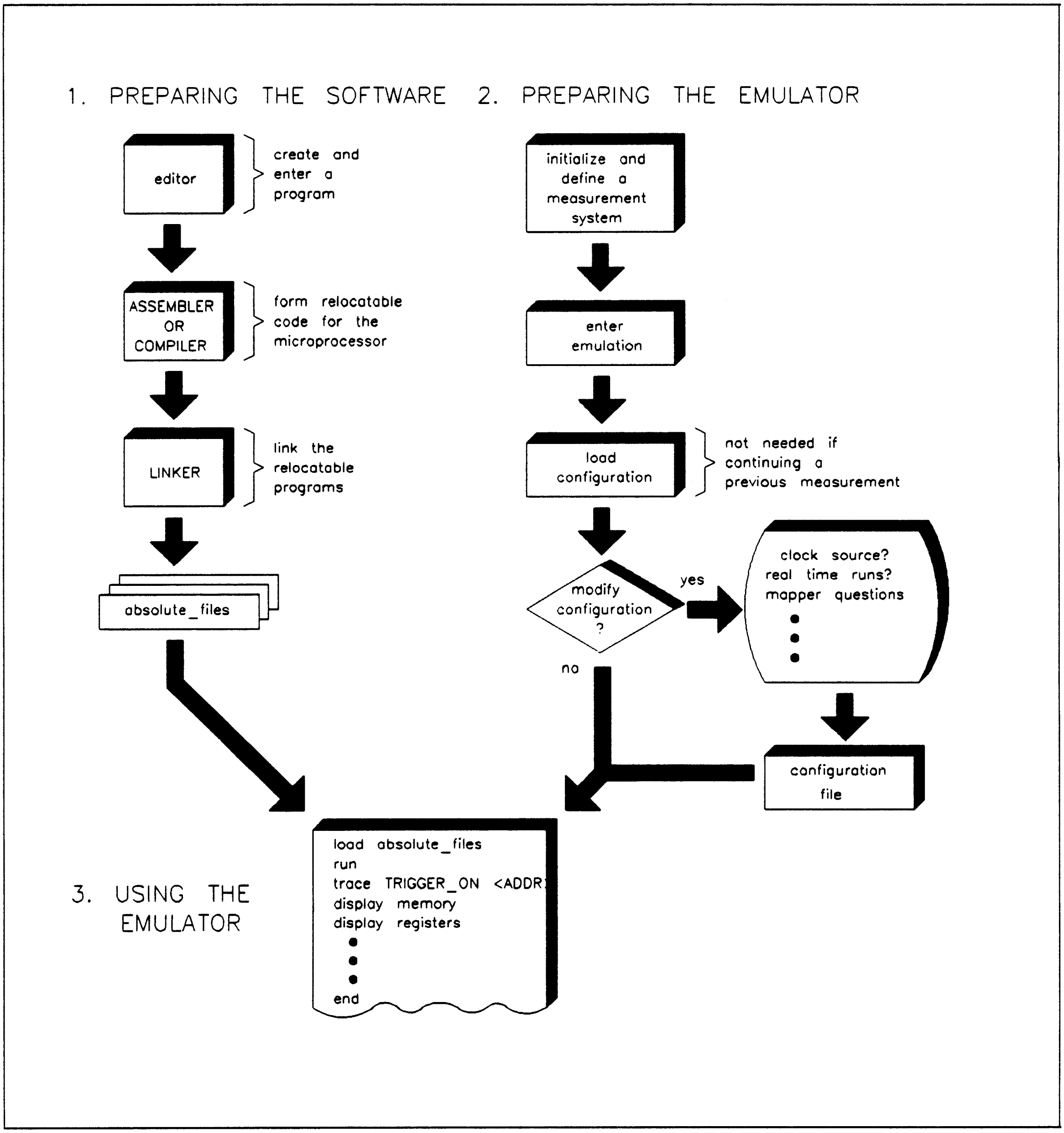


Figure 1-1. Steps to Using the Emulator

Notes

2

Emulation Command Syntax

Overview

This chapter:

- Describes the syntax conventions used in this manual.
- Summarizes the emulation commands.
- Gives a detailed description of each emulator command.

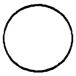
Conventions

Here are the conventions used in the command syntax diagrams:

This symbol shows a command keyword that you enter by pressing a softkey. The keyword appears as it would in the command line, which may not be the same as the softkey label.

Rectangular boxes contain either a prompt showing parameters that you must enter or a reference to another syntax diagram. Softkey prompts are enclosed by the "<" and ">" symbols and are shown exactly as they appear on the softkey label. **--EXPR--** and **--SYMB--** are prompts that access "expression help" softkeys. You can return to the normal set of emulation softkeys by pressing **--NORMAL--**. This chapter includes syntax diagrams for **--EXPR--** and **--SYMB--**.

References to additional syntax diagrams may be shown in upper or lower case characters without delimiters.



Circles denote operators and delimiters used in expressions and command lines.

Whenever keywords entered from softkeys appear in text or examples, they are shown in bold type, for example: **copy.** Parameters entered from the keyboard are shown in standard type.

Command Summary

Table 2-1 summarizes the emulation commands. The remainder of this chapter gives detailed descriptions of each command.

Note

Some command parameters shown in the following syntax diagrams may not be available during emulation. The softkeys that are available depends on how you configure the emulator.

For example, if you have not configured simulated I/O to be used during your session and you enter the command:

display

the **sim_io** softkey will not be shown. Your answers to other emulation configuration questions also affect the softkey labels. Only softkeys that are enabled for your emulation configuration are displayed.

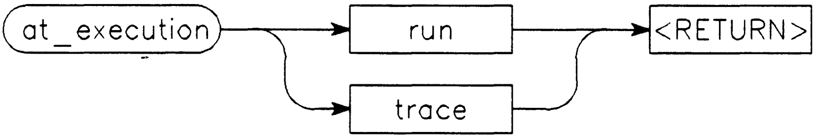
Table 2-1. Emulation Command List

at_execution
break
copy display
copy global_symbols
copy local_symbols
copy memory
copy mmu_mappings
copy mmu_tables
copy registers
copy software_breakpoints
copy trace*
copy trace_specification*
copy help**
display global_symbols
display local_symbols
display memory
display mmu_mappings
display mmu_tables
display registers
display simulated_io
display source_file
display software_breakpoints
display trace*
display trace_specification*
end
execute
halt
help**
load configuration
load memory
load symbols
load trace_specification
modify analysis
modify configuration
modify keyboard_to_simio
modify memory
modify register
modify software_breakpoints
performance_measurement
reset
run
set
set analysis*
set bnc_ports*
set source
set symbols
set <VAR>
set WIDTH
step
store
trace*
wait**

* These commands are described in the *Analysis Reference Manual for 32-Bit Microprocessors.*
**Hidden commands; not displayed on the softkeys. Must be typed in at the keyboard.

at_execution

Syntax



Function You use **at_execution** to prepare a run or trace command for execution. Use this command with the **execute** command. If the processor is not reset, **at_execution run** causes a break from your program, and initializes the monitor to the default address or to the specified address. An **execute** command then starts the run. The **execute** command removes the run specification, which cannot be repeated without respecifying the run.

at_execution trace initializes the trace hardware with the given trace specification. An **execute** command will start the trace. A trace specification is not removed. It can be repeated without another **at_execution trace** command. **at_execution trace** and **at_execution run** can be used with a single **execute** command that begins the run, the trace, and any other analyzers that are connected to the intermodule bus (IMB).

A **trace** command cancels an **at_execution trace** command. A **run** or **step** command cancels an **at_execution run** command. The **at_exec** softkey label is displayed only with multiple module systems.

Default Value none

Example

```
at_execution run from START
at_execution trace TRIGGER_ON a= 1234h
```

See Also:

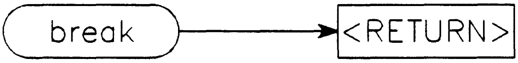
- Execute syntax (in this chapter)

- Emulation configuration (chapter 4 in the *68030 Emulator User's Guide*).

- Operating In the Measurement System (in the *HP 64000-UX User's Guide*).

break

Syntax



Function

Break diverts the processor from execution of your program to the emulation monitor program.

The **break** softkey is not displayed if the emulation monitor is not loaded.

Default Value

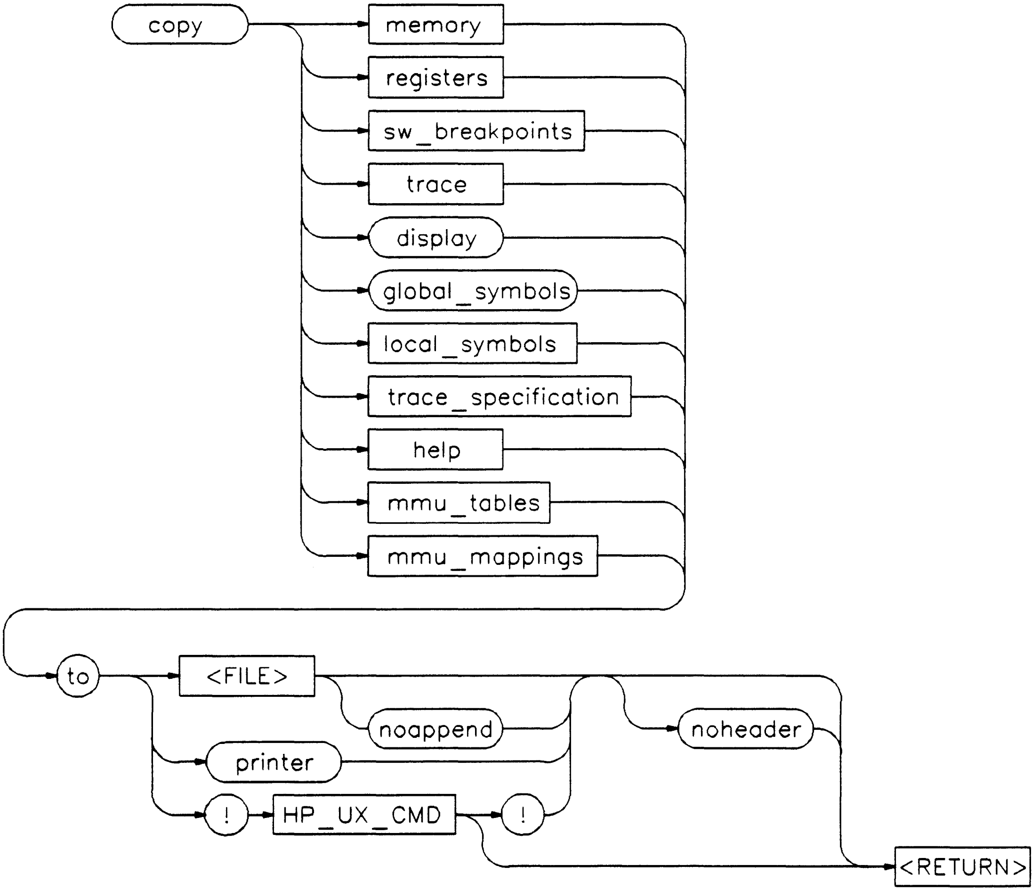
none

Example

```
break
```

copy

Syntax



Function The **copy** command copies selected information to your system printer, to a listing file, or pipes it to an HP-UX filter.

Default Values

Depending on the information selected, defaults may be the options selected for the previous execution of the **display** command.

Parameters

display	**display** copies the information displayed on the screen to the selected destination.
<FILE>	**<FILE>** prompts you for the name of the listing file where the specified information is to be copied.
global_symbols	**global_symbols** copies the global symbols from the symbol database to the selected destination.
help	**help** copies the contents of the emulation help files to the selected destination. The keyword "help" is not available on the softkeys. You must type it from the keyboard. After you type help, the emulation help filenames are displayed on the softkeys.
HP-UX CMD	**HP-UX CMD** represents an HP-UX filter or pipe. The output of the copy command will be routed to this command. HP-UX commands must be preceded by an exclamation point (!). An exclamation point following the HP-UX command continues command line execution after execution of the HP-UX command. HP-UX commands that are shell intrinsics don't affect emulation.
local_ symbols_in	**local_symbols_in** copies a list of local symbols in a specified source file to the selected destination. Local symbols are those that are children of the specified symbol. That is, they are defined in that symbol's scope. See the --SYMB-- syntax pages and the *HP 64000-UX System User's Guide* for more information.

memory — **memory** copies the contents of memory to the selected destination.

mmu_mappings — **mmu_mappings** copies the logical-to-physical address mappings for a particular root pointer.

mmu_tables — **mmu_tables** copies the mapping information for a particular logical address.

noappend — **noappend** overwrites any existing file specified by <FILE> with the copied information. If **noappend** is not specified, the default operation is to append the copied information to the end of an (existing) file.

noheader — **noheader** copies the information without headings.

printer — **printer** specifies your system printer as the destination device for the **copy** command. Before you can specify printer as the destination device, you must first define PRINTER as a shell variable.

```
$ PRINTER=lp
$ export PRINTER
```

registers — **registers** copies the contents of the various register sets to the selected destination.

software_ breakpoints — **software_breakpoints** copies the current software breakpoint table to the selected destination.

to — **to** specifies the destination of the copied information. **to** must be included in the command line.

trace — **trace** copies some or all of the current trace listing to the selected destination.

trace_specification

trace_specification copies some or all of the trace specification to the selected destination.

!

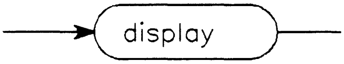
The exclamation point is the delimiter for HP-UX commands.

An exclamation point must precede all HP-UX commands. A trailing exclamation point to return to command line execution is optional.

If an exclamation point is part of the HP-UX command, a backslash (\) must be used to escape the exclamation point (\!).

copy display

Syntax



Function

The **copy display** command copies the information currently displayed on the screen.

Default Value

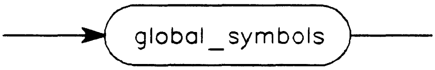
none

Examples

```
copy display to printer
copy display to trcfile1
```

copy global_symbols

Syntax



Function

The **copy global_symbols** command copies the global symbols defined for the current absolute file. Global symbols are those that are declared as global (XDEF) in the source file. They include procedure names, variables, constants, and file names. The listing will include the symbol name, logical address, segment containing the symbol, and the symbol's offset from the start of the segment.

Default Value

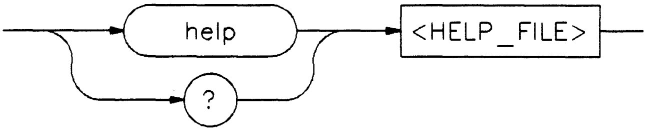
None

Examples

```
copy global_symbols to printer
copy global_symbols to symbols noheader
```

copy help

Syntax



Function The **copy help** command copies the contents of a specified help file. The help command is not displayed on the softkeys. You must type it at the keyboard. You can substitute a question mark (?) for the keyword **help** in the command string.

Default Value none

Examples

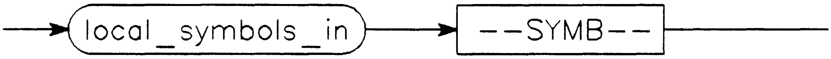
```
copy help system_commands to printer
copy ? trace to trc_cmd
```

Parameters

HELP_FILE — HELP_FILE is the name of the help file you want to copy. After you type **help** from the keyboard, the help file names are available on the softkeys.

copy local_symbols_in

Syntax



Function The **copy local_symbols_in** command copies the local symbols in a specified source file or scope, their addresses, their relative segment, and offset. Local symbols are the children of the symbol specified by --SYMB--. That is, they are defined within that symbol.

Default Value none

Example

```
copy local_symbols_in sample(module) to
printer
```

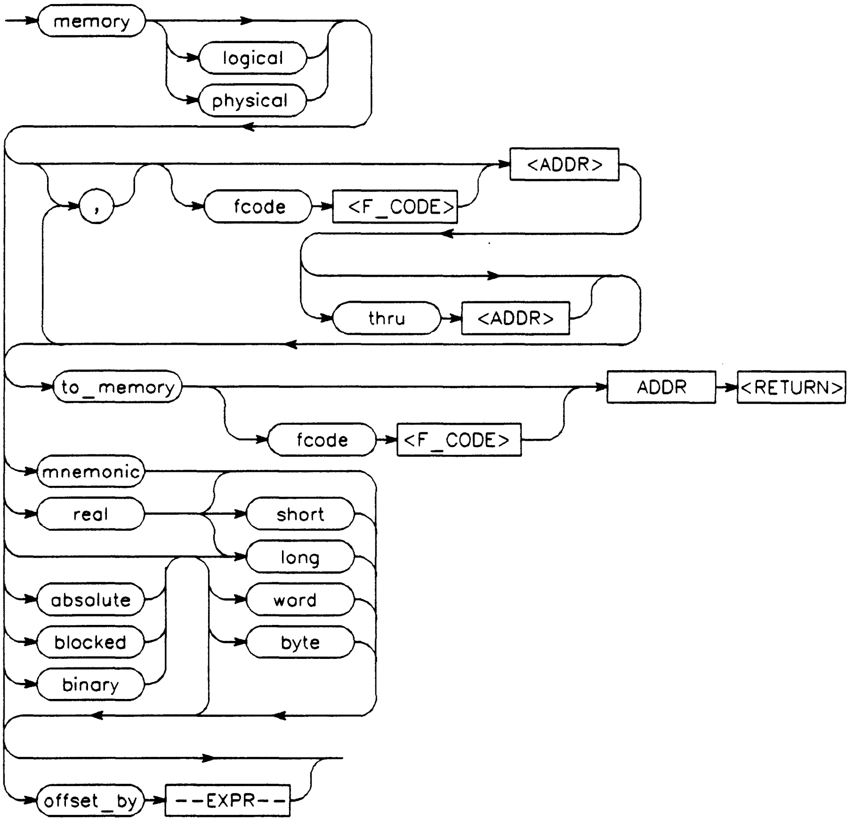
Parameters

--SYMB--	--SYMB-- represents the source file that contains the local symbols to be listed. See the --SYMB-- syntax diagram.

See Also See the --SYMB-- syntax pages and the *HP 64000-UX System User's Guide* for more information on symbols.

copy memory

Syntax



Function The **copy memory** command copies the contents of the specified memory location or series of locations.

Memory can be copied to the system printer, to a listing file, to another area of memory, or piped to an HP-UX filter. When copying to another area of memory, the destination memory locations must be in target RAM or emulation memory mapped as RAM or ROM.

The memory contents can be listed either in mnemonic, binary, hexadecimal, or real number format. In addition, the memory addresses can be listed offset by a value, allowing easy comparison of the information to the program assembly listing.

Note

The **copy memory** command works only when you are using the background monitor. The foreground monitor does not implement the memory accesses needed to support this command.

Default Values

Initial values are the same as specified by the command **display memory 0 blocked words offset_by 0**.

Defaults are to values specified in the previous **display memory** command.

Examples

```
copy memory fcode SUPER_PROG START thru
START+3ffH mnemonic to printer

copy memory fcode SUPER_DATA 0 thru 100H ,
fcode SUPER_PROG START thru START+5  blocked
long to memlist

copy memory fcode SUPER_PROG 1000 thru 13ffh
to_memory fcode USER_PROG 2000h
```

Parameters

absolute

absolute formats the memory listing in a single column.

<ADDR>

<ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a memory address or offset value. See the --EXPR-- syntax diagram.

binary

binary copies the contents of memory locations as binary values.

blocked

blocked formats the memory listing in multiple columns.

fcode

fcode enables you to specify a function code with the address expression as part of the memory access specification.

<F_CODE>

<F_CODE> is a prompt for the function code. You can specify the function code as a number or as a defined function code mnemonic on the softkeys.

logical

logical treats the address specification as a logical address.

long

long copies the memory values as long word values.

When used with the **real** parameter, **long** copies memory in a 64-bit real number format.

mnemonic

mnemonic formats the memory listing in assembly language instruction mnemonics with associated operands. When you select mnemonic format, specify a starting address that corresponds to the first word of an opcode. This ensures that the listed mnemonics are correct.

offset_by

offset_by enables you to specify an offset that is subtracted from each absolute address before listing the addresses and the corresponding memory contents. You can select an offset value (--EXPR--) such that each module in a program appears to start at address 0000H. The memory contents listing will then appear similar to the assembly or compiler listing.

physical

physical treats the address specification as a physical address.

real

real formats the memory values in the listing as real numbers.

short

Use **short** with real to format memory values as 32-bit real numbers.

thru

thru specifies that a range of memory locations be copied.

to_memory

to_memory copies a block of memory to another location in memory.

words

words copies the memory listing as word values.

,

A comma (,) appearing immediately after **memory** in the command line appends the current **copy memory** command to the preceding **display memory** command. The data specified in both commands is copied to the destination selected in the current command. The current command specifies the data format.

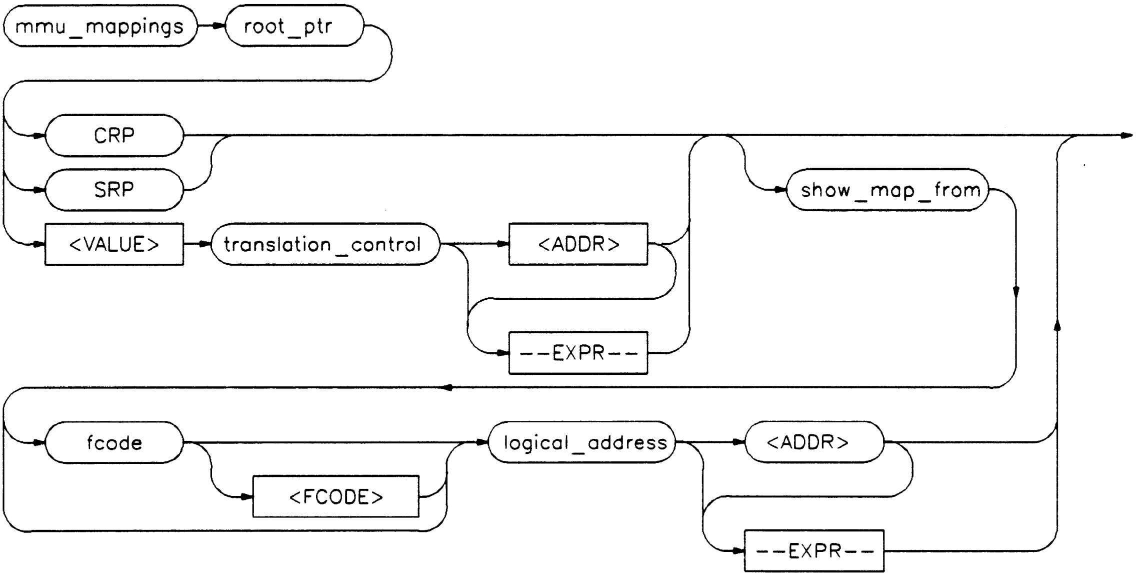
The comma is also used as a delimiter between values when specifying multiple memory addresses.

Function codes are an important part of the memory access specification, with the address expression. The function code (if stated explicitly) precedes the associated address expression, and may be specified as a number or a predefined function code mnemonic (for example: SUPER_PROG, USER_DATA).

Memory configuration allows different modes for function codes: they may be enabled (full use of function codes), disabled (no use of function codes), or partially disabled (only PROGRAM/DATA spaces are recognized). If the function codes are disabled (even partially), the unused function code bits are masked off and ignored during the memory access.

copy mmu_mappings

Syntax



Function

The **copy mmu_mappings** command copies the overall logical-to-physical address mapping information for a particular root pointer to the specified destination.

Note

The **copy mmu_mappings** command works only when you are using the background monitor. The foreground monitor does not implement the memory accesses needed to support this command.

Default Values

None.

Examples

```
copy mmu_mappings root_ptr CRP to printer

copy mmu_mappings root_ptr
08000000 2000f4000h translation_control
8c0c440h to map_table.txt

copy mmu_mappings root_ptr CRP show_map_from
fcode USER_DATA logical_address 2000H to
mymap
```

Parameters

<ADDR> This prompts you to enter an address expression. See the --EXPR-- syntax diagram for details.

CRP CPU Root Pointer.

fcode Use this to specify the function code you want to begin your mmu_mappings list, or mmu_tables display.

<FCODE> Prompts you to enter a function code, either as a number or as a function code shown on the softkeys.

logical_address The address in logical (virtual) memory space.

root_ptr Use this to introduce the source of the root pointer descriptor.

show_map_from Use this to specify the logical address (with or without function code) where you want your mapping list or tables display to begin.

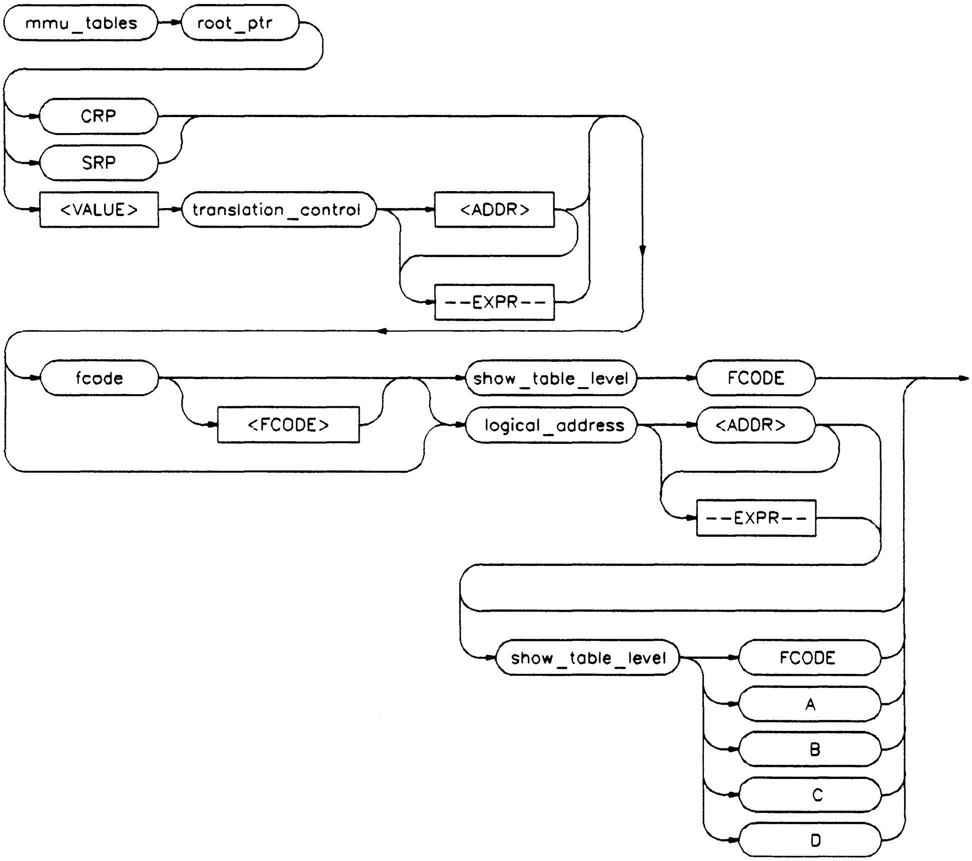
SRP Supervisor Root Pointer

translation_ control Use this to specify a value for the translation control register. This is required when you specify a value for the root pointer.

<VALUE> Root pointer value to be used instead of the CRP or SRP. You also must specify the value of the translation control register when you specify a root pointer value.

copy mmu_tables

Syntax



Function The **copy mmu_tables** command copies the mapping information for a particular logical address to a specified destination.

Note

The **copy mmu_tables** command works only when you use the emulator's background monitor. The foreground monitor doesn't implement the memory accesses required to support this command.

Default Values

none

Examples

```
copy mmu_tables root_ptr CRP logical address
0 to mytables
```

```
copy mmu_tables root_ptr CRP fcode USER_DATA
logical_address 02000000H show_table_level
FCODE to printer noheader
```

```
copy mmu_tables root_ptr 011B
translation_control 82CF5000H
logical_address 02000000H show_table_level B
```

Parameters

<ADDR> This prompts you to enter an address expression. See the --EXPR-- syntax diagram for details.

CRP CPU Root Pointer.

fcode Use this to specify the function code you want to begin your mmu_mappings list, or mmu_tables display.

<FCODE> Prompts you to enter a function code, either as a number or as a function code shown on the softkeys.

logical_address The address in logical (virtual) memory space.

root_ptr Use this to introduce the source of the root pointer descriptor.

show_table_level

Use this to specify the table level that you want to examine in detail in your mmu_tables copy.

SRP

Supervisor Root Pointer

translation_control

Use this to specify a value for the translation control register. This is required when you specify a value for the root pointer.

<VALUE>

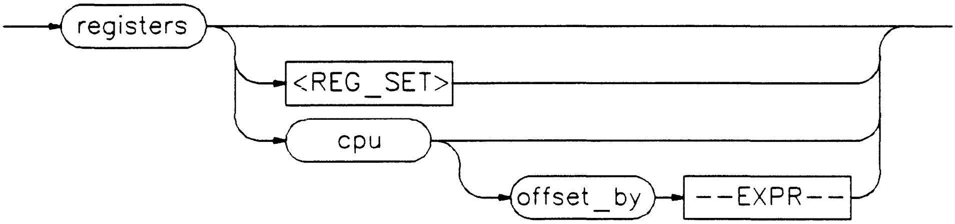
Root pointer value to be used instead of the CRP or SRP. You also must specify the value of the translation control register when you specify a root pointer value.

Note

See the *Motorola MC68030 Enhanced 32-Bit Microprocessor User's Manual* for more information on root pointers.

copy registers

Syntax



Function

The **copy registers** command copies the current contents of the processor/coprocessor's various register sets. This process does not occur in real time. You must configure the emulator for nonreal-time run mode to list registers while the processor is running.

You can supply a number to offset the CPU program counter from the actual value. This allows easy comparison of some registers to the assembled listing.

When you specify a custom coprocessor, the coprocessor register set is appended to the CPU register set listing.

Default Values

Initially cpu registers with 0 offset. After that, defaults to the last **copy registers** command specification.

Examples

```
copy registers mmu to reglist
copy registers cpu offset_by 10f0h to printer
```

Parameters

--EXPR-- --EXPR-- is a combination of numeric values, symbols, operators, and parentheses specifying an offset value to be subtracted from the program counter. See the --EXPR-- syntax diagram.

offset_by

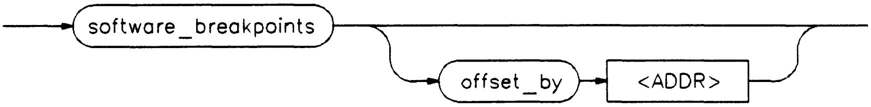
offset_by specifies an offset to subtract from the actual **cpu** program counter address before the program counter value is copied. You can select the offset value (--EXPR--) such that the program counter address will match the current instruction's address in the assembler or compiler listing.

<REG_SET>

<REG_SET> specifies the name of the register set to be displayed. You can select a register set name from the softkeys. All custom coprocessor names defined in your custom register specification file are displayed. The name **cpu** specifies the 68030's internal cpu registers. The name **fpu** is reserved for the emulator's internal 68881 floating point processor, if used.

copy software_ breakpoints

Syntax



Function

The **copy software_breakpoints** command copies the currently defined software breakpoints and their status. If you're continuing emulation from a previous session, then the listing includes any previously defined breakpoints. The column marked "status" shows whether the breakpoint is pending or inactivated. A pending breakpoint forces the processor to enter the emulation monitor on execution of that breakpoint. Breakpoints that were defined as one_shot are listed as inactivated after they are executed. Entries that show an inactive status can be reactivated by executing the **modify software_breakpoints set** command.

Default Value

none

Examples

```
copy software_breakpoints to printer
copy software_breakpoints offset_by 0f000h
to breaklist noheader
```

Parameters

<ADDR> <ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying an offset from the listed software breakpoint address. See the --EXPR-- syntax diagram.

offset_by **offset_by** allows you to offset the listed software breakpoint address value from the breakpoint's actual address. The system subtracts the offset from the breakpoint's actual address, making the listed address match that given in the assembler or compiler listing.

copy trace

Function

The **copy trace** command enables you to copy some or all of the current trace listing to the selected destination.

See the *Analysis Reference Manual for 32-Bit Microprocessors* for a detailed description of the **copy trace** command.

copy trace_specification

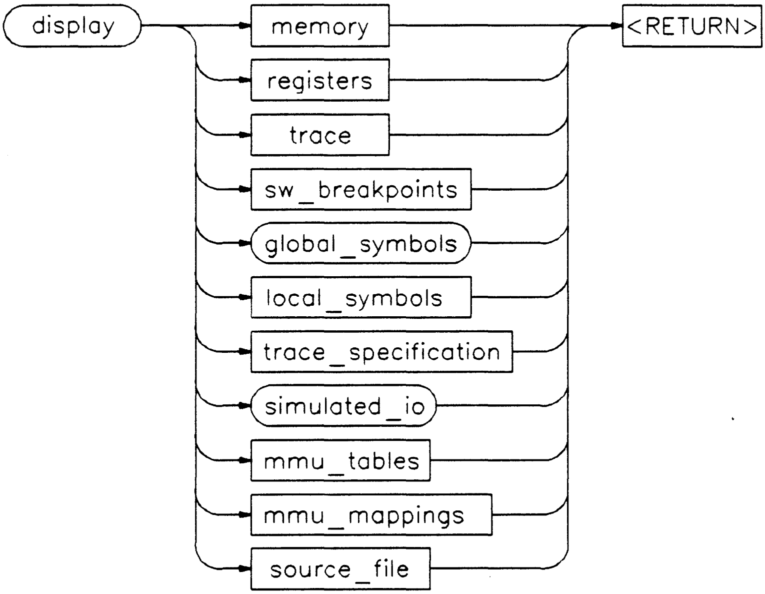
Function

The **copy trace_specification** command enables you to copy some or all of your trace specification to the selected destination.

See the *Analysis Reference Manual for 32-Bit Microprocessors* for a detailed description of the **copy trace_specification** command.

display

Syntax



Function The **display** command displays selected information on your workstation screen. You can use the **UP** and **DOWN** cursor keys, the **NEXT** and **PREV** keys, and sometimes, the **LEFT** and **RIGHT** cursor keys to view the displayed information. **<Ctrl>F** and **<Ctrl>G** may be used to scroll the screen left and right by header columns.

Default Values Depending on the information selected, defaults may be the options selected for the previous execution of the **display** command.

Parameters

global_symbols **global_symbols** displays a list of all global symbols in memory.

local_symbols_in

local_symbols_in displays a list of local symbols defined in a specified symbol. Local symbols are those that are children of the specified symbol. See the --SYMB-- syntax pages and the *HP 64000-UX System User's Guide* for more information.

memory

memory displays the contents of memory.

mmu_mappings

mmu_mappings displays the logical-to-physical address mappings for a particular root pointer.

mmu_tables

mmu_tables displays the mapping information for a particular logical address.

registers

registers displays the contents of the microprocessor registers.

simulated_io

simulated_io displays the data being written to the simio display buffer.

software_breakpoints

software_breakpoints displays the current software breakpoint table.

source_file

source file displays the content of the source file you specify.

trace

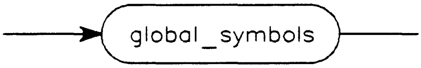
trace displays the current trace listing.

trace_specification

trace_specification displays your current trace specification, starting at optionally defined points.

display global_symbols

Syntax



Function

The **display global_symbols** command displays the global symbols defined for the current absolute file. Global symbols are those that are declared as global (XDEF) in the source file. They include procedure names, variables, constants, and file names. When you use this command, the listing will include the symbol name, logical address, segment containing the symbol, and the symbol's offset from the start of the segment.

Default Value

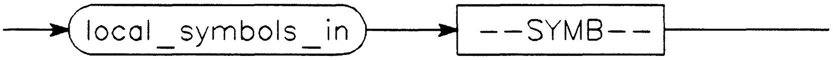
none

Example

```
display global_symbols
```

display local_symbols_in

Syntax



Function The **display local_symbols_in** command displays the local symbols in a specified symbol, which may include various combinations of source file and scope. Displayed information includes symbol addresses plus relative segment and offset.

Default Value none

Example

```
display local_symbols_in towers(module)
```

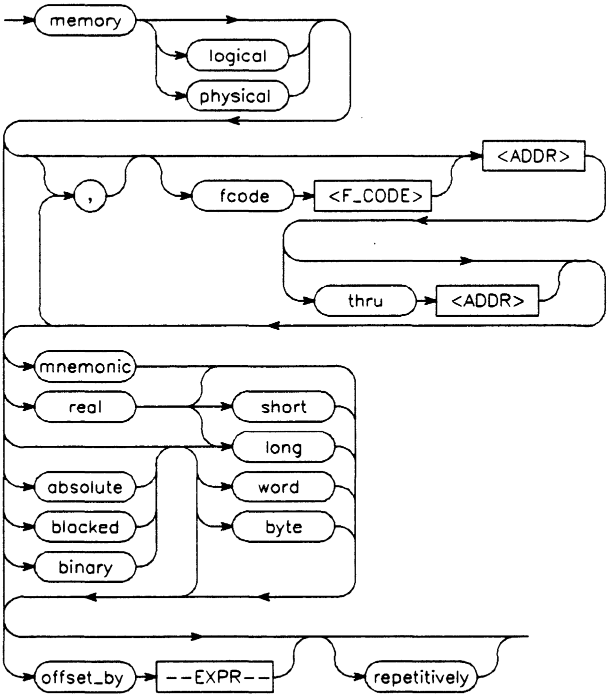
Parameters

--SYMB-- --SYMB-- represents the symbol that contains the local symbols to be listed.

See Also See the --SYMB-- syntax pages and the *HP 64000-UX System User's Guide* for more information on symbols and their scoping.

display memory

Syntax



Function The **display memory** command displays the contents of the specified memory location or series of locations. The memory contents can be listed in mnemonic, binary, hexadecimal, or real number format. Memory addresses can be listed offset by a value that allows the information to be easily compared to the program listing.

Default Values Initial values are the same as specified by the command **display memory** 0 **blocked word offset_by** 0.

Default for "logical" or "physical" addresses is "logical" to start, then the same as requested in a previous command.

Other defaults are to values specified in previous display memory command.

Each memory access command has a separate function code default to be used when a function code is valid, but not explicitly specified.

Examples

```
display memory fcode SUPER_PROG START
mnemonic offset_by 1f00h
display memory fcode USER_DATA 0 thru 100H ,
fcode USER_PROG START thru START+5 blocked
word
```

Parameters

absolute	**absolute** formats the memory listing in a single column.
<ADDR>	**<ADDR>** is a combination of numeric values, symbols, operators, and parentheses specifying a memory address or memory offset value. See the --EXPR-- syntax diagram.
binary	**binary** displays the contents of memory locations as binary values.
blocked	**blocked** formats the memory listing in multiple columns.
fcode	**fcode** enables you to specify a function code with the address expression as part of the memory access specification.
<F_CODE>	**<F_CODE>** is a prompt for the function code. The function code may be specified as a number or as a defined function code mnemonic on the softkeys.

logical

logical specifies that the address space to be displayed is logical space.

long

long displays the memory values as long word values.

When used with the **real** parameter, **long** displays memory in a 64-bit real number format.

mnemonic

mnemonic formats the memory listing in assembly language instruction mnemonics with associated operands. You should specify a starting address that corresponds to the first word of an opcode to ensure that the listed mnemonics are correct.

offset_by

offset_by allows you to specify an offset that is subtracted from each absolute address before listing the addresses. You can choose the offset value (--EXPR--) so that each module in a program appears to start at address 0000H. The memory contents listing will then appear similar to the assembly or compiler listing.

physical

physical specifies that the address space to be displayed is physical space.

real

real formats the memory values in the listing as real numbers.

repetitively

repetitively continuously updates the memory listing displayed on your screen.

short

Use **short** with real to list memory values as 32-bit real numbers.

thru

thru enables you to specify that a range of memory locations be displayed. The amount of information displayed at once is restricted by your display terminal. Use the **UP** and **DOWN**

cursor keys, and the **NEXT** and **PREV** keys to view additional memory locations.

words

words displays the memory listing as word values.

,

A comma (,) appearing immediately after **memory** in the command line will append the current **display memory** command to the preceding **display memory** command. The data specified in both commands is displayed. The data is formatted as specified in the current command.

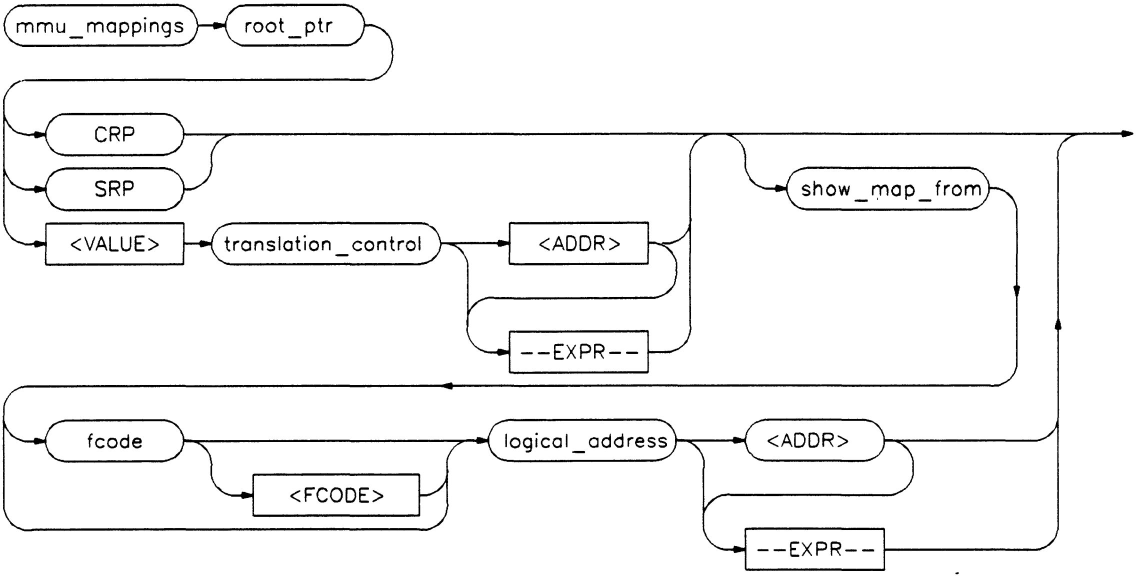
The comma is also used as a delimiter between values when specifying multiple memory addresses.

Function codes are an important part of the memory access specification, with the address expression. The function code (if stated explicitly) precedes the associated address expression, and may be specified as a number or a predefined function code mnemonics (for example: SUPER_PROG, USER_DATA).

Memory configuration allows different modes for function codes: they may be enabled (full use of function codes), disabled (no use of function codes), or partially disabled (only PROGRAM/DATA spaces are recognized). If the function codes are disabled (even partially), the unused function code bits are masked off and ignored during the memory access.

display mmu_mappings

Syntax



Function

The **display mmu_mappings** command displays the overall logical-to-physical address mapping information for a particular root pointer.

Note

The **display mmu_mappings** command works only when you are using the background monitor. The foreground monitor does not implement the memory accesses needed to support this command.

Default Values

None.

Examples

```
display mmu_mappings root_ptr CRP
display mmu_mappings root_ptr
0800000002000f4000h translation_control
8c0c440h
display mmu_mappings root_ptr CRP
show_map_from fcode USER_DATA
logical_address 2000H
```

Parameters

<ADDR> This prompts you to enter an address expression. See the --EXPR-- syntax diagram for details.

CRP CPU Root Pointer.

fcode Use this to specify the function code you want to begin your mmu_mappings list, or mmu_tables display.

<FCODE> Prompts you to enter a function code, either as a number or as a function code shown on the softkeys.

logical_address The address in logical (virtual) memory space.

root_ptr Use this to introduce the source of the root pointer descriptor.

show_map_from Use this to specify the logical address (with or without function code) where you want your mapping list or tables display to begin.

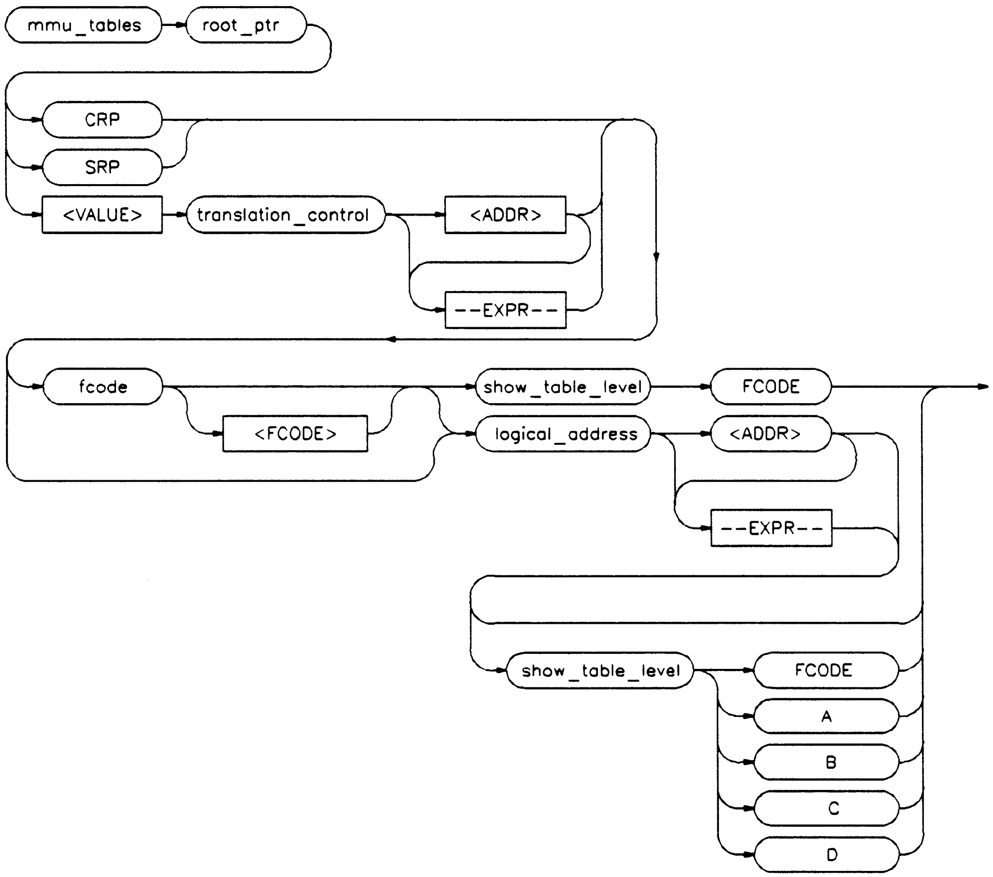
SRP Supervisor Root Pointer

translation_control Use this to specify a value for the translation control register. This is required when you specify a value for the root pointer.

<VALUE> Root pointer value to be used instead of the CRP or SRP. You also must specify the value of the translation control register when you specify a root pointer value.

display mmu_tables

Syntax



Function The **display mmu_tables** command displays the mapping information for a particular logical address.

Note

The **display mmu_tables** command works only when you use the emulator's background monitor. The foreground monitor doesn't implement the memory accesses required to support this command.

Default Values

none

Examples

```
display mmu_tables root_ptr CRP logical
address 0

display mmu_tables root_ptr CRP fcode
USER_DATA logical_address 02000000H
show_table_level FCODE

display mmu_tables root_ptr 011B
translation_control 82CF5000H
logical_address 02000000H show_table_level B
```

Parameters

<ADDR> This prompts you to enter an address expression. See the --EXPR-- syntax diagram for details.

CRP CPU Root Pointer.

fcode Use this to specify the function code you want to begin your **mmu_mappings** list, or **mmu_tables** display.

<FCODE> Prompts you to enter a function code, either as a number or as a function code shown on the softkeys.

logical_address The address in logical (virtual) memory space.

root_ptr Use this to introduce the source of the root pointer descriptor.

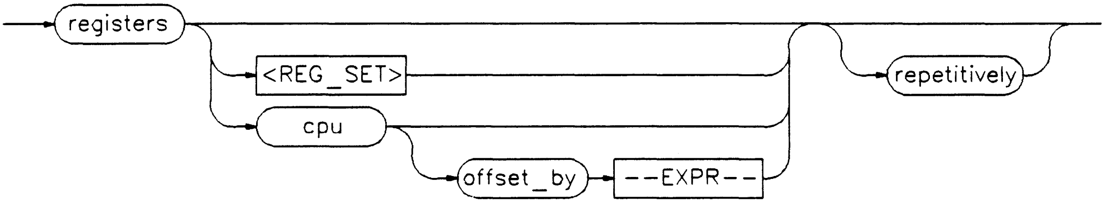
show_table_level	Use this to specify the table level that you want to examine in detail in your mmu_tables copy.
SRP	Supervisor Root Pointer
translation_control	Use this to specify a value for the translation control register. This is required when you specify a value for the root pointer.
<VALUE>	Root pointer value to be used instead of the CRP or SRP. You also must specify the value of the translation control register when you specify a root pointer value.

Note

See the *Motorola MC68030 Enhanced 32-Bit Microprocessor User's Manual* for more information on root pointers.

display registers

Syntax



Function The **display registers** command displays the current contents of the processor/coprocessor's various register sets. If a step has just been executed, the mnemonic of the last instruction is also displayed. This process does not occur in real time. You must configure the emulator for nonreal-time run mode if you want to display registers while the processor is running.

The displayed value of the CPU program counter can be offset from the actual value by a number that allows the register information to be easily compared to the assembler listing.

When you specify a custom coprocessor, the coprocessor register set is appended to the CPU register set listing.

Default Values Offset is initially 0. After that, offset is the previous value.

Example

```
display registers cpu
```

Parameters

--EXPR-- **--EXPR--** is a combination of numeric values, symbols, operators, and parentheses specifying an offset value to be subtracted from the program counter. See the --EXPR-- syntax diagram.

offset_by

offset_by enables you to specify an offset that is subtracted from the actual **cpu** program counter address before the program counter value is displayed. You can choose the offset value (--EXPR--) such that the program counter address will match the current instruction's address in the assembler or compiler listing.

<REG_SET>

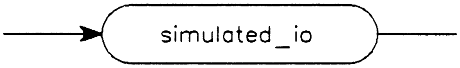
<REG_SET> specifies the register set to be displayed. You can select the register set names from softkeys. All custom coprocessor names defined in your custom register specification file are displayed. The name **cpu** specifies that the 68030's internal cpu registers be displayed. The name **fpu** is reserved for the emulator's internal 68881 floating point processor, if used.

repetitively

repetitively continuously updates the register listing displayed on your screen.

display simulated_io

Syntax



Function

The **display simulated_io** command displays the information being written to the simulated I/O display buffer. See the *HP 64000-UX Simulated I/O Reference Manual* and chapter 9 of the *68030 Emulator User's Guide* for detailed information.

Default Value

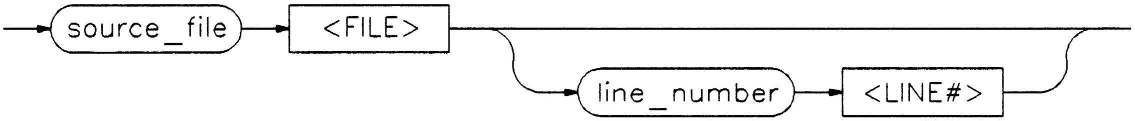
none

Example

```
display simulated_io
```

display source_file

Syntax



Function

The display source_file command displays the content of the source file you specify on screen. You can use the roll and paging keys of your keyboard to look at any desired area of your source file, or you can specify any line in your source file to begin the display. You cannot modify the content of your source file through this display.

Default Values

display source_file <FILE> **line_number** 0.

Example

```
display source_file test.c
display source_file keyboard.c line_number 40
```

Parameters

<FILE>

<FILE> is a prompt for the name of the file to be displayed. Be sure to include the file-type identifier (".c" in the example) with the name of the file.

line_number

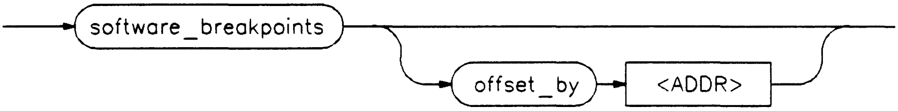
line_number enables you to specify a line number in your source file where the display should begin. The line number you specify will be at the top of the source-file display.

<LINE#>

<LINE#> is a prompt for the source_file line number where you want to begin the display.

display software _breakpoints

Syntax



Function The **display software_breakpoints** command displays the currently defined software breakpoints and their status. If you're continuing emulation from a previous session, then the listing includes any previously defined breakpoints. The column marked "status" shows whether the breakpoint is pending or inactivated. A pending breakpoint forces the processor to enter the emulation monitor on execution of that breakpoint. Breakpoints that were defined as one_shot are listed as inactivated after they are executed. Entries that show an inactive status can be reactivated by executing the **modify software_breakpoints set** command.

Default Value none

Examples

```
display software_breakpoints
display software_breakpoints offset_by 1000H
```

Parameters

<ADDR> <ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying an offset value for the breakpoint address. See the --EXPR-- syntax diagram.

offset_by **offset_by** allows you to offset the listed software breakpoint address value from the breakpoint's actual address. The system subtracts the offset value from the breakpoint's actual address. This can make the listed address match that given in the assembler or compiler listing.

display trace

Function The **display trace** command enables you to display some or all of the current trace listing.

See the *Analysis Reference Manual for 32-Bit Microprocessors* for a detailed description of the **display trace** command.

display trace_specification

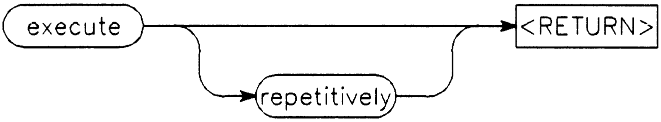
Function

The **display trace_specification** command enables you to display some or all of your trace specification.

See the *Analysis Reference Manual for 32-Bit Microprocessor*s for a detailed description of the **display trace_specification** command.

execute

Syntax



Function The **execute** command starts a trace measurement. The **execute** softkey label is replaced with the **halt** softkey label when a measurement is in progress. If emulation is participating in a system measurement through cross-triggered analysis or the emulation start function (**at_execution run** or **at_execution trace**), then the system measurement is begun. Otherwise, the **execute** command is not available.

You can continuously repeat a measurement by using the **execute repetitively** command. This restarts the current measurement after each completion, until you give the **halt** command. The **execute** command starts all modules participating in a system measurement when issued from any one of the modules. If an emulator is started as part of a measurement, it continues running and cannot be restarted by subsequent executions unless an **at_execution run** command is reissued.

The **execute** softkey is displayed only when multiple modules are present in a system and some IMB interaction is requested (cross-triggered analysis or emulation start function).

Examples

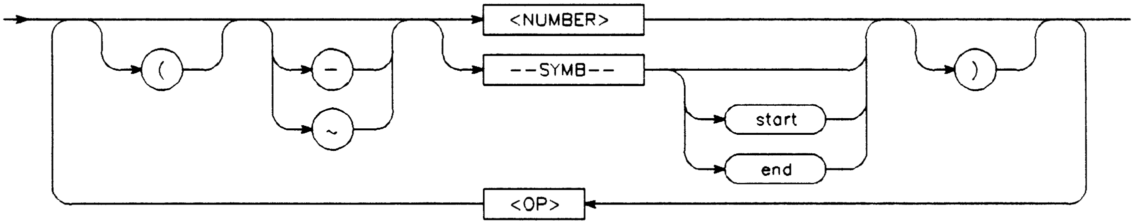
```
execute
execute repetitively
```

See Also:

- **at_execution** command (in this chapter)
- Emulation configuration (chapter 4 of the *68030 Emulator User's Guide)*
- The “Operating in the Measurement System” section of the *HP 64000-UX User's Guide.*

--EXPR--

Syntax



Function

An expression is a combination of numeric values, symbols, operators, and parentheses specifying an address, data, status, or any of several other value types used in the emulation commands.

Default Value

none

Examples

```
05fxh (not valid for all commands)
DISP_BUF + 5
SYMB_TBL + (OFFSET / 2)
START
prog(module): line 15 end
```

Parameters

<NUMBER>   <NUMBER> is a numeric value in binary, octal, decimal, or hexadecimal base.

<OP>

<OP> is an algebraic or logical operand. <OP> can be (in order of precedence):

mod	modulo
*	multiplication
/	division
&	logical AND
+	addition
-	subtraction
|	logical OR

--SYMB--

--SYMB-- is a symbolic reference to an address or address range, file, or other value. See the --SYMB-- syntax pages and the *HP 64000-UX System User's Guide* for more information on symbols.

( )

Parentheses may be used in expressions to alter evaluation precedence or add clarity. For every opening parenthesis, there must be a closing parenthesis.

-

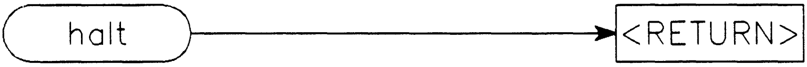
Algebraic negation (minus)

~

logical negation (NOT)

halt

Syntax



Function

The **halt** command stops the current measurement and turns off the **repetitively** option. When you give the **halt** command, some or all systems involved may have completed their measurement. The **halt** softkey is displayed only during a trace, or during an execution (in the place of the **execute** softkey).

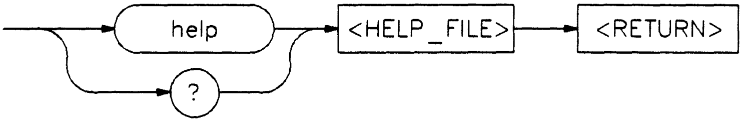
The **halt** command affects measurements caused by both **trace** and **execute** commands. If emulation is entered with a measurement in progress, the **halt** command will stop that measurement even if emulation is not interacting in the measurement.

Example

```
halt
```

help

Syntax



Function The **help** command enables you to request information about system and emulation features during your emulation session. Typing "help" or "?" from the keyboard displays softkey labels that list the areas on which you may receive help. Press the softkey for the command in which you are interested, and then press the **return** key. The system displays the information on the screen using the HP-UX **more** utility.

The help command is not displayed on the softkeys. You must type it on the keyboard. You can substitute a question mark (?) for the keyword "help" in the command string.

Default Value none

Examples

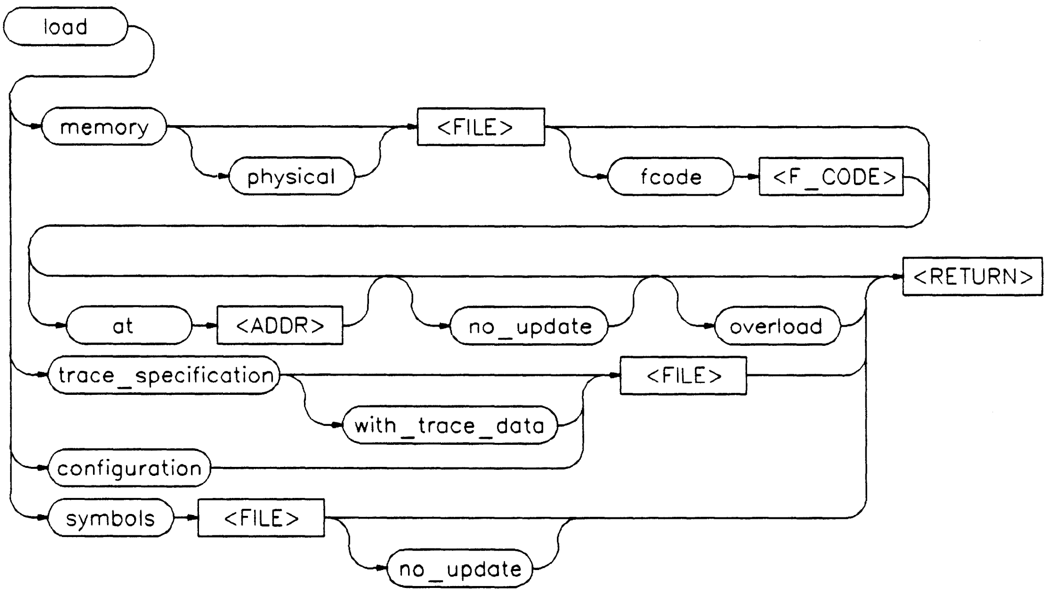
```
help system_commands
? trace
```

Parameters

HELP_FILE	HELP_FILE is the name of the help file you wish to display. After you type "help" from the keyboard, the help file names can be entered from the softkeys.

load

Syntax



Function

The **load memory** command transfers absolute code from the host system disc into target system RAM or emulation memory. The memory configuration map and the address specified during linking determine the destination of the absolute code. The map is defined during emulator configuration.

You can load the absolute code at a location other than the address specified during linking by using the **at <ADDR>** parameter. When using **at <ADDR>**, the absolute code is loaded in memory beginning at the specified address. For example, if you specify "**at** 2000h," you are effectively specifying an offset of +2000h for your code.

Note

Don't use the **at <ADDR>** feature if your code uses absolute addressing. Absolute addresses and symbol values in your program are not modified. This may cause run-time errors or unexpected behavior.

The **load configuration** command reloads a previously saved emulation configuration.

The **load trace_specification** command reloads a previous trace specification. If you saved the trace specification with trace data, you can use the **display** command to access and display the previously stored trace data. You can execute the previously stored trace specification using the **trace again** or **execute** commands.

Default Value

For the **load memory** command, all memory is in the default function code space.

Examples

```
load memory physical  sort
load configuration config3
load trace_specification trace3
```

Parameters

at — **at** lets you load absolute code at a location other than the address specified during linking.

configuration — **configuration** loads a configuration file created by a **modify configuration** command.

fcode — **fcode** enables you to specify a function code with the address expression as part of the memory access specification.

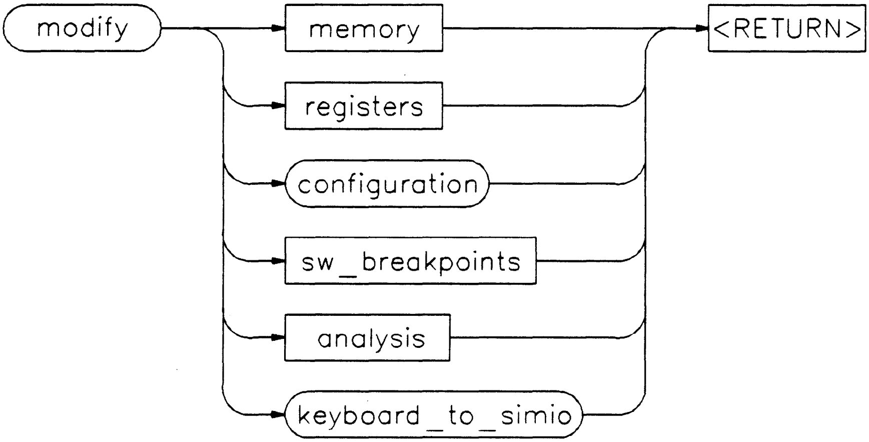
<FILE> — <FILE> is the pathname of the absolute file to be loaded from the system disk into target system RAM, emulation memory, or the trace

memory (.TR files are assumed) containing a previously stored trace specification and trace listing.

<F_CODE>	<F_CODE> is a prompt for the function code. You may specify the function code as a number or as a defined function code mnemonic on the softkeys.
memory	**memory** loads an absolute file into emulation or target memory.
no_update	This option suppresses rebuilding of the SRU symbol database when you load an absolute file. Normally, the symbol database is rebuilt if the absolute file has changed since the last time the symbol database was updated.
overload	**overload** forces loading of the absolute file and suppresses warning messages. Normally, if you load a file with symbols that have already been loaded, and the symbols were used in a trace specification, you will receive a warning that existing symbols will be lost.
physical	**physical** specifies that the address space to be loaded is physical space.
symbols	**symbols** loads a symbol database with the specified filename.
trace_ specification	**trace_specification** loads a trace file that was generated using the **store trace** command.
with_trace_data	**with_trace_data** loads the trace data with the trace specification, if the trace data was stored.

modify

Syntax



Default Value none

Parameters

analysis	**analysis** allows you to change any part of your analysis trace specification or trace command.
configuration	**configuration** enables you to review and modify (if necessary) the current emulation configuration.
memory	**memory** allows you to modify the contents of selected memory locations.
registers	Use **registers** to modify the contents of one or more of the various register sets.

software_ breakpoints — **software_breakpoints** sets or clears software breakpoints used with the emulator break function.

trace_command — **trace_command** recalls the last trace command for editing.

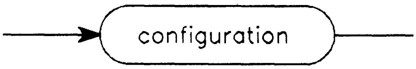
modify analysis

Function The **modify analysis** command lets you change any part of your analysis trace specification or **trace** command.

See the *Analysis Reference Manual for 32-Bit Microprocessors* for a detailed description of the **modify analysis** command.

modify configuration

Syntax



Function

The **modify configuration** command enables you to review and edit the current emulation configuration. Each configuration question is presented with the response previously entered. You can select the previous response by pressing the **return** key, or modify it as necessary and then enter it by pressing the **return** key.

Default Value

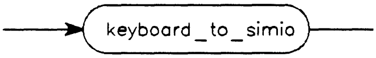
none

Example

```
modify configuration
```

modify keyboard_to_simio

Syntax



Function

The **modify keyboard_to_simio** command activates the keyboard to interact with your program through the HP 64000-UX simulated I/O software. When you activate the keyboard for simulated I/O, its normal interaction with emulation is disabled. The emulation softkeys are blanked and the single softkey **suspend** is displayed on your screen. Press **suspend** and then the **return** key to deactivate keyboard simulated I/O and return the keyboard to normal emulation mode. Refer to the *HP 64000-UX Simulated I/O Reference Manual* and chapter 9 of the *68030 Emulation User's Guide* for detailed information about simulated I/O.

Default Value

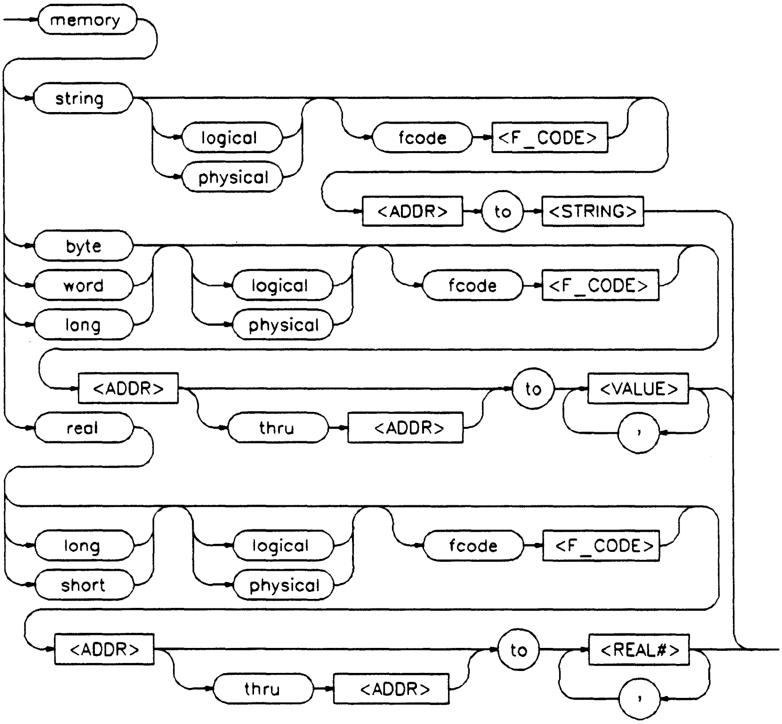
none

Example

```
modify keyboard_to_simio
```

modify memory

Syntax



Function

The **modify memory** command enables you to modify the contents of selected memory locations. You can modify the contents of each memory location in a series to an individual value or the contents of all locations in a memory block to a single or repeated sequence of values.

Function codes are an important part of the memory access specification, along with the address expression. The function code (if stated explicitly) precedes the associated address expression. You can specify the function code as a number or one of the defined function code mnemonics (for example: SUPER_PROG, USER_DATA).

Note

If the specified address range is too small to contain the new data, the emulator will modify as many locations as required to contain the new data, beginning with the starting address you specified.

New data value lists will be repeated as needed to fill the specified address ranges. Any remaining values will modify address locations after the last address in the specified address range.

Default Values

Each memory access command has a separate function code default that is used when a function code is valid, but not explicitly specified.

Examples

```
modify memory word logical fcode SUPER_DATA
00A0h to 1234h
modify memory word fcode USER_DATA DATA1 to
0E3h , 01h , 08h
modify memory real long TEMP to 0.5532E-8
modify memory string BUFFER to "This string"
```

Parameters

<ADDR>

<ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See the --EXPR-- syntax diagram.

byte

byte specifies that the memory values be modified as byte values.

fcode

fcode enables you to specify a function code with the address expression as part of the memory access specification.

<F_CODE>

<F_CODE> is a prompt for the function code. You can specify the function code as a number

or as a defined function code mnemonic on the softkeys.

logical

logical specifies that the address space to be modified is in logical space.

long

long specifies that the memory values be modified as long word values.

When used with the **real** parameter, **long** specifies that memory be modified as a 64-bit real number value.

physical

physical specifies that the address space to be modified is in physical space.

real

real specifies that the memory values be modified as real number values.

<REAL #>

<REAL #> prompts you to enter a value in real number format.

short

short is used with real to specify that memory values be modified as 32-bit real number values.

string

string enables you to modify a series of byte locations to the ASCII values contained in a string literal value.

<STRING>

<STRING> is a prompt for a quoted ASCII string literal such as "This is a string".

thru

thru enables you to specify that a range of memory locations be modified.

to

to enables you to specify the values to which the selected memory locations will be changed.

word

word specifies that the memory locations be modified as word values.

, commas (,) are delimiters between values when modifying multiple memory addresses.

Description

You can modify a series of memory locations by specifying the address of the first location in the series to be modified (--EXPR--) and the list of the values (--EXPR--) to which the contents of that location and the succeeding locations are to be changed. The first value listed replaces the contents of the specified memory location, the second value replaces the contents of the next location in the series, and so on until the list has been exhausted. If only one number or symbol is specified, only that address is modified. When more than one value is listed, the value representations must be separated by commas.

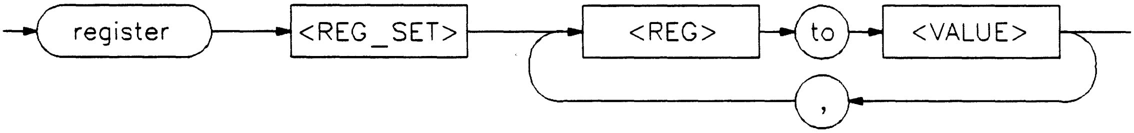
You can modify an entire block of memory such that the contents of each location in the block is changed to the single specified value, or to a single or repeated sequence. Do this by entering the limits of the memory block to be modified (--EXPR-- thru --EXPR--) and the value or list of values (--EXPR--, ... , --EXPR--) to which the contents of all locations in the block are to be changed.

Function codes are an important part of the memory access specification, with the address expression. The function code (if stated explicitly) precedes the associated address expression, and may be specified as a number or one of the defined function code mnemonics (for example: SUPER_PROG, USER_DATA).

Memory configuration allows different modes for function codes: they may be enabled (full use of function codes), disabled (no use of function codes), or partially disabled (only PROGRAM/DATA spaces are recognized). If the function codes are disabled (even partially), then the unused function code bits are masked off and ignored during the memory access.

modify register

Syntax



Function

Use the **modify register** command to modify the contents of one or more registers in the processor/coprocessor's register set. The entry for <REG> determines the register to modify.

You can't modify registers when the emulator is restricted to real-time runs. Break to the monitor to access the registers.

Default Value

none

Examples

```
modify registers cpu D0 to 9H
modify registers cpu A0 to 1001b , A1 to
1023h
```

Parameters

<REG>

<REG> represents the name of the register to be modified. The softkey labels display the possibilities for <REG>.

<REG_SET>

<REG_SET> specifies the name of the register set to be modified. Select the register set names from softkeys. All custom coprocessor names defined in your custom register specification file are displayed. The name **cpu** specifies the 68030's internal cpu registers. The name **fpu** is

reserved for the emulator's internal 68881 floating point processor, if used.

to

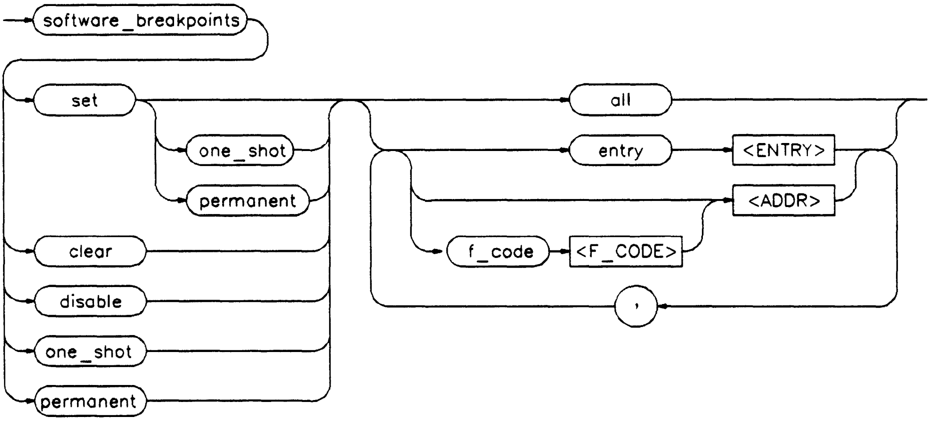
to enables you to specify the values to which the selected registers will be changed.

<VALUE>

<VALUE> is a combination of numeric values, symbols, operators, and parentheses specifying a register value. See the --EXPR-- syntax diagram.

modify software_ breakpoints

Syntax



Function

Software breakpoints enable the emulator to "break on execution" of an instruction at a specified address. You can specify any valid address (number, label or expression) as a breakpoint. Valid addresses identify the first word of valid instructions.

Resume program operation after the breakpoint by using either a **run** or **step** command.

Default Values

none

Examples

```
modify software_breakpoints clear fcode
USER_PROG 1099h , 1234h
modify software_breakpoints set fcode
SUPER_PROG one_shot  LOOP1END , LOOP2END
modify software_breakpoints clear entry 1
modify software_breakpoints disable entry 2
```

Parameters

<ADDR>

<ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a software breakpoint address. See the --EXPR-- syntax diagram.

all

If used with the **set** parameter, **all** reactivates all breakpoint entries (sets them to pending). If used with the clear parameter, **all** clears all entries and restores the original values of the memory locations. **all** also enables you to disable all entries or to change all entries to one-shot or permanent mode.

clear

clear clears the specified breakpoint address <ADDR> and restores the original contents of the memory location.

disable

disable deactivates the selected breakpoint entry.

<F_CODE>

<F_CODE> is a prompt for the function code. If you use a function code, it must be specified using a predefined function code mnemonic from the softkeys.

one_shot

one_shot sets the breakpoint for one execution. On execution, the breakpoint is deactivated and the original content of the memory location is restored. Also, use **one_shot** to modify the mode of existing entries.

permanent

permanent sets the breakpoint until you clear or disable it. The breakpoint can be repeatedly executed. **permanent** is also used to modify the mode of existing entries.

set

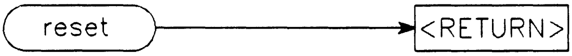
set adds software breakpoints to your program.

,

Commas (,) are delimiters between specified breakpoint values.

reset

Syntax



Function

The **reset** command suspends target system operation and establishes initial operating parameters, such as reloading control registers. The reset signal is latched when the reset command is executed and is released by the run command.

When the processor is released from reset by a run command, one of two operations will occur, depending on the answer to the reset_to_monitor configuration question:

- Reset_to_monitor enabled: the processor will reset into the monitor, ignoring any user-defined reset vector.

- Reset_to_monitor disabled: the processor will vector into the reset handler defined by the user reset vector.

Default Value

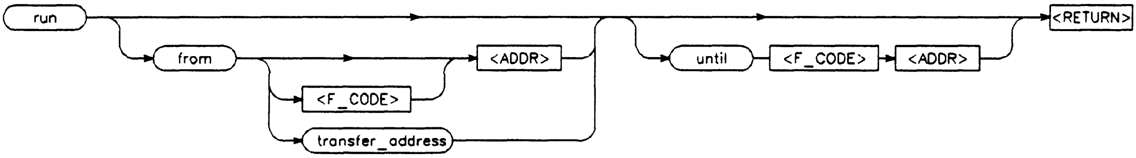
none

Example

```
reset
```

run

Syntax



Function

If the processor is in a reset state, **run** will release the reset, and if a "**from**" address is specified the processor is started at that address. If the processor is running in the monitor, the **run** command causes the processor to exit into your program. The program can either be run from a specified address (--EXPR--), from the address currently stored in the processor's program counter, or from a label specified in the program.

The program will run until the **until** address is encountered and then break to the monitor. The **until** <ADDR> specification sets a software breakpoint at the requested address.

Default Value

If you omit the address (--EXPR--) option, the emulator will begin program execution at the address in the processor's program counter.

Examples

```
run
run from 810H
run from USER_STATE START until LOOP_1
run until SUPERVISOR_STATE LOOP_1
```

Parameters

<ADDR> <ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a

memory address. See the --EXPR-- syntax diagram.

<F_CODE>

<F_CODE> is a prompt for the function code. If used, the function code must be specified using a predefined function code mnemonic from the softkeys.

from

from is used to specify the address from which program execution is to begin.

transfer_address

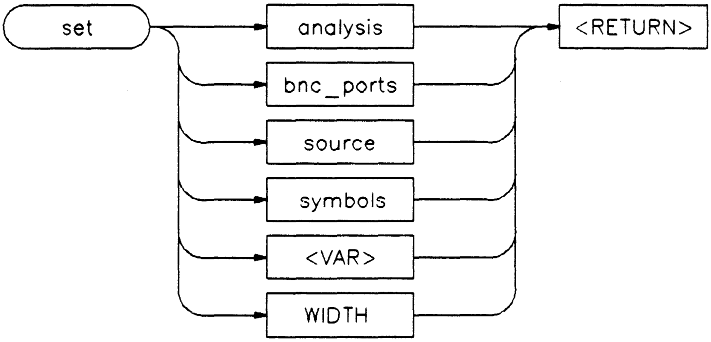
transfer_address is the starting address of the program you loaded into emulation or target memory. The transfer_address is defined in the linker map.

until

until is used in defining a software breakpoint on which to break execution of your program.

set

Syntax



Function Allows you to set parameters for analyzer measurements, cross-trigger configuration, symbol and source display, and system environment.

Default Values See the syntax pages for the set options.

Parameters

analysis — Allows you to set the analyzer configuration and operating characteristics. Refer to the *32-bit Analysis Reference Manual* for more information on these options.

bnc_ports — You use this to set up the cross-trigger configuration (for measurements involving other instruments). Refer to the *32-bit Analysis Reference Manual* for more information on these options.

source — You use this option to control source code display. See the pages that follow for more information on this option.

symbols

You use this option to control symbol display. See the pages that follow for more information on this option.

<VAR>

You use this to set system environment parameters. This option is described on the following pages.

WIDTH

You can use this option to control the display width allocated to showing information in the address and mnemonic columns, and in the lines of source-file information. Remember to "set source/symbols on" before setting display width.

set analysis

Function The **set analysis** command lets you change your prestore or GLOBAL_CONTEXT specification, set your trigger_position and analysis break condition, or change your analysis softkey interface.

See the *Analysis Reference Manual for 32-Bit Microprocessors* for a detailed description.

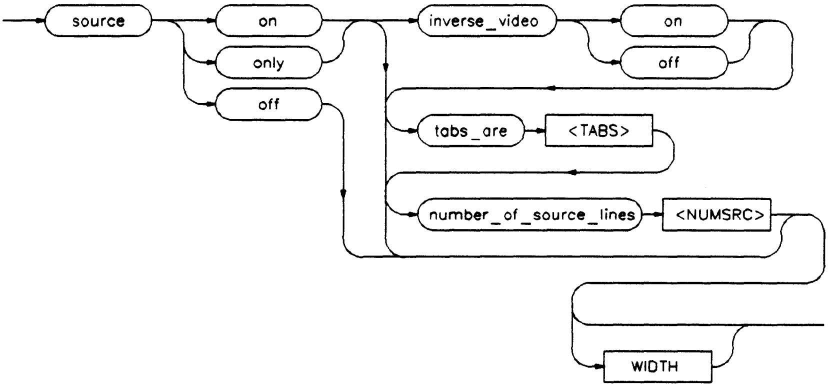
set bnc_ports

Function The **set bnc_ports** command lets you change any portion of your bnc port configuration.

See the *Analysis Reference Manual for 32-Bit Microprocessors* for a detailed description of the **set bnc_ports** command.

set source

Syntax



Function You use this command to control display of source lines in emulator measurement displays, such as the analyzer trace.

Examples

```
set source on inverse_video on tabs_are 2
```

Default Values The default display format parameters are the same as set by the command:

```
set source off symbols on
```

Parameters

inverse video

off	This displays source lines in normal video.
on	This highlights the source lines on the screen (dark characters on light background) to differentiate the source lines from other data on the screen.

number_of_source_lines

This allows you to specify the number of source lines displayed for the actual processor instructions with which they correlate. Only source lines up to the previous actual source line will be displayed. Using this option, you can specify how many comment lines are displayed preceding the actual source line. The default value is 5.

<NUMSRC>

This prompts you for the number of source lines to be displayed. Values in the range 1..50 may be entered.

source

off

This option prevents inclusion of source lines in the trace and memory mnemonic display lists.

on

This option displays source program lines preceding actual processor instructions with which they correlate. This enables you to correlate processor instructions with your source program code.

only

This option displays only source lines.

tabs_are

This option allows you to define the number of spaces inserted for tab characters in the source listing.

<TABS>

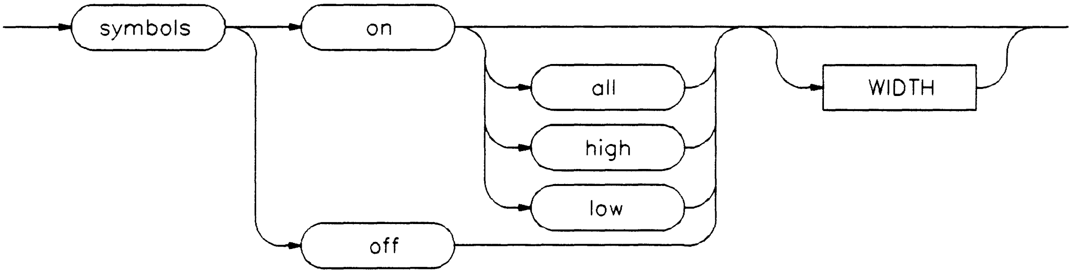
Prompts you for the number of spaces to use in replacing the tab character. Values in the range 2..15 may be entered.

WIDTH

Refer to set WIDTH page for this syntax.

set symbols

Syntax



Function You use this command to control symbol display in various emulator measurement screens, such as the analyzer trace.

Examples

```
set symbols on
```

Default Values The default display format parameters are the same as set by the command:

```
set source off symbols on
```

Parameters

symbols

off	This prevents symbol display.
on	This displays symbols.
high	Displays only high level symbols, such as those available from a compiler. See the HP 64000-UX System User's Guide for a detailed discussion of symbols.

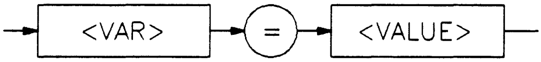
low — Displays only low level symbols, such as those generated internally by a compiler, or an assembly symbol.

all — Displays all symbols.

WIDTH — Refer to set WIDTH page for this syntax.

set <VAR>

Syntax



Function

You use the set <VAR> command to define system environment variables for use within an emulation session. For example, if you enter the command:

```
set x = /users/guest/test
```

then, at any later time, "$x" may be used as an alias for "/users/guest/test." For example:

```
load memory $x/myfile
```

A <VALUE> that contains embedded spaces must be enclosed within quotation marks. Also, any HP-UX environment variables that were defined and exported prior to the emulation session may be used.

Default Values

none

Examples

```
set emuldir = /users/<yourlogon>/emul683k
set dispmem = "display memory 1000h"
```

Allowing you to use:

```
cd $emuldir
$dispmem blocked word
```

Parameters

<VAR>

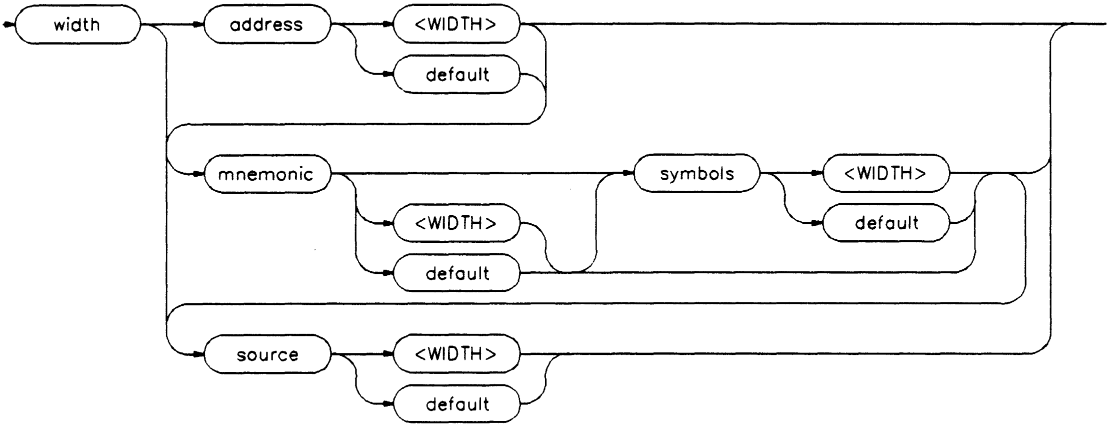
<VAR> specifies an environment variable name, consisting of a string of letters, and/or digits.

<VALUE> **<VALUE>** is the alias assigned to the environment variable (<VAR>), consisting of a string of letters, and/or digits.

= Equal (=) signs indicate that the environment variable **<VAR>** is to be set to **<VALUE>**.

set WIDTH, SOURCE WIDTH, SYMBOLS WIDTH

Syntax



Function

You use this command to set widths of columns of information displayed on screen. The widths you specify will apply in trace list displays and memory displays.

Examples

```
set width address 24 mnemonic 45 symbols 22
set symbols on high width mnemonic symbols 30
set source only width source 78
```

Default Values

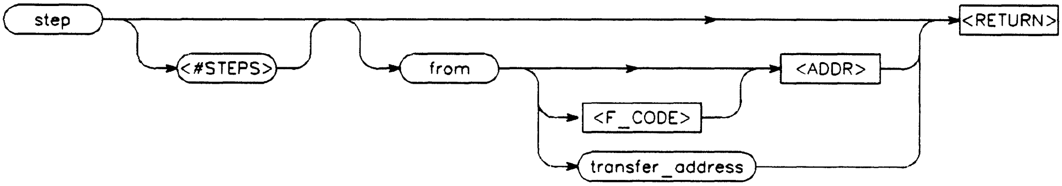
address default = 12
mnemonic default = 55
symbols default = 16
source default = 67

Parameters

address	This option allows you to specify the width allocated to the display of address information.
default	This specifies use of the default value for the associated parameter.
mnemonic	This option allows you to specify the width allocated to display of information in the mnemonic column.
source	This option allows you to specify the display width allocated to source-file lines.
symbols	This option allows you to specify the width allocated to the display of symbols.
<WIDTH>	This prompts you to enter a display width for the associated option.

step

Syntax



Function

The **step** command causes the emulation processor to execute a specific number of instructions. This allows sequential analysis of program execution. The contents of the processor registers, the contents of trace memory, and the contents of emulation or target memory can be displayed after each step command completes.

Default Values

If no value is entered for <NUMBER> of times, only one instruction is executed each time you press the **return** key. Multiple instructions also can be executed by holding down the **return** key.

If the from address (--EXPR-- or transfer_address) option is omitted, stepping begins at the next address.

Examples

```
step
step from fcode SUPERVISOR_STATE 810h
step 20 from fcode USER_STATE 0A0h
```

Parameters

<ADDR> <ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See the --EXPR-- syntax diagram.

<F_CODE>

<F_CODE> is a prompt for the function code. If used, the function code must be specified using a predefined function code mnemonic from the softkeys.

from

from is used to specify the address from which program stepping is to begin.

<NUMBER>

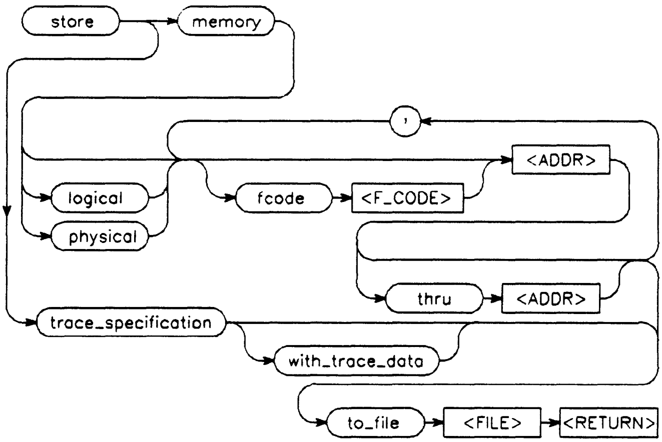
<NUMBER> determines how many instructions will be executed by the step command. The number of instructions to be executed can be entered in binary (B), decimal (D), octal (O or Q), or hexadecimal (H) notation.

transfer_address

transfer_address is the starting address of the program you loaded into emulation or target memory. The transfer_address is defined in the linker map.

store

Syntax



Function Use the store command to store the contents of specific memory locations into an absolute file (.X file), or to store the trace specification, with or without trace data, into a trace file (.TR file).

Default Value None

Examples

```
store memory logical fcode USER_PROG 800h
thru 20ffh to_file temp2
store trace_specification to_file trclst
```

Parameters

--EXPR-- --EXPR-- is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See the --EXPR-- syntax diagram.

fcode

fcode enables you to specify a function code with the address expression as part of the memory access specification. This parameter is valid only if different function code ranges are defined in the memory map.

<F_CODE>

<F_CODE> is a prompt for the function code. The function code may be specified as a number or as a defined function code mnemonic on the softkeys.

<FILE>

<FILE> is a prompt for the identifier for the absolute file or trace file in which data is to be stored.

The store command creates a new file having the specified name when there is no absolute file on the disc with that name. If the file already exists, the system asks whether the old file is to be deleted. If the response is yes, the new file replaces the old one. If the response is no, then the **store** command is canceled and no data is stored. The transfer address of the absolute file is set to zero.

logical

logical specifies that the selected memory locations to be stored are in logical space.

memory

memory stores the selected memory locations in the specified file.

physical

physical specifies that the selected memory locations to be stored are in physical space.

thru

thru enables you to store memory ranges.

to_file

to_file must be used in the **store memory** command to separate the memory location specifications from the file identifier (<FILE>).

trace_specification — **trace_specification** stores the current trace specification in the specified file.

with_trace_data — **with_trace_data** stores the trace data with the trace specification.

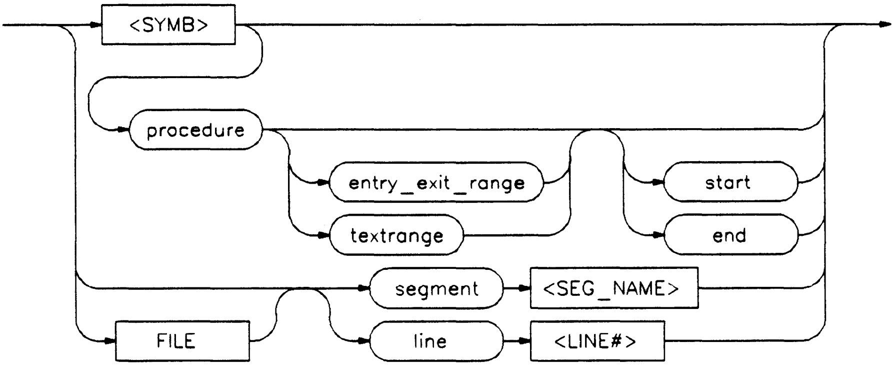
, — Commas (,) are used to separate memory expressions in the command line.

--SYMB--

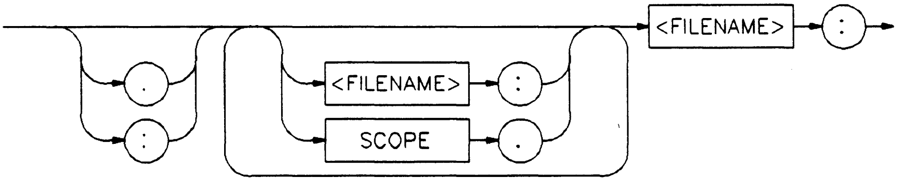
This parameter is a symbolic reference to an address, address range, file, or other value.

Syntax

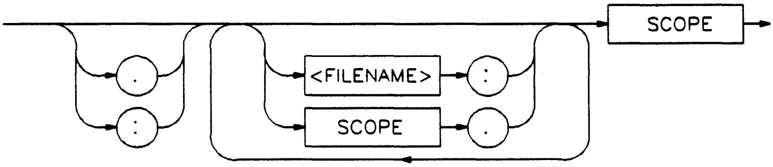
--SYMB--



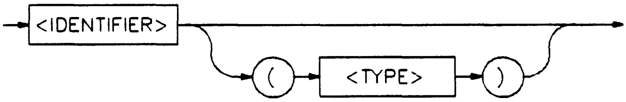
FILE



<SYMB>



SCOPE



Note

If no default file or module was defined by executing the command **display local_symbols_in --SYMB--**, or with the **cws** command, a source file name (**<FILE>**) or module name must be specified with each local symbol in a command line.

Function

Symbols may be:

- Combinations of paths, filenames, and identifiers defining a scope, or referencing a particular identifier or location (including procedure entry and exit points).
- Combinations of paths, filenames, and line numbers referencing a particular source line.
- Combinations of paths, filenames, and segment identifiers identifying a particular PROG, DATA or COMN segment or a user-defined segment.

The Symbolic Retrieval Utilities (SRU) handle symbol scoping and referencing. These utilities build trees to identify unique symbol scopes.

If you use the SRU utilities to build a symbol database before entering the emulation environment, the measurements involving a particular symbol request will occur immediately. If you then change a module and reenter emulation without rebuilding the symbol database, the emulation software rebuilds the changed portions of the database as necessary.

Further information regarding the SRU and symbol handling is available in the *HP 64000-UX System User's Guide*. Also refer to that manual for information on the **HP64KSYMBPATH** environment variable.

Default Value

The last symbol specified in a **display local_symbols_in --SYMB--** command, or with the **cws** command, is the default symbol scope. The default is "none" if no current working symbol was set in the current emulation session.

You also can specify the current working symbol by typing the **cws** command on the command line and following it with a symbol name. The **pws** command displays the current working symbol on the status line.

Display memory mnemonic also can modify the current working symbol.

Parameters

end
: The last address associated with the given procedure or range.

entry_exit_range
: This is the range of addresses associated with all code for the given procedure. May differ from **textrange**, which is all addresses associated with the starting and ending text points of the procedure.

<FILENAME>
: This is an HP-UX path specifying a source file. If no file is specified, and the identifier referenced is not a global symbol in the executable file that was loaded, then the default file is assumed (the last absolute file specified by a display local_symbols_in command). A default file is only assumed when other parameters (such as **line**) in the **--SYMB--** specification expect a file.

line
: This specifies that the following numeric value references a line number in the specified source file.

<LINE#>
: Prompts you for the line number of the source file.

<IDENTIFIER>
: Identifier is the name of an identifier as declared in the source file.

procedure

procedure indicates that you want to use the address range of a procedure with the given symbol name.

SCOPE

Scope is the name of the portion of the program where the specified identifier is defined or active (such as a procedure block). Scope also may refer to a module name in IEEE-695 file format.

segment

This shows that the following string specifies a standard segment (such as PROG, DATA, or COMN) or a user-defined segment in the source file.

<SEG_NAME>

Prompts you for entry of the segment name.

start

The first address associated with the given procedure or range.

textrange

Represents the range of code addresses associated with the given procedure. May be different than **entry_exit_range,** especially if the compiler generates subroutine code that is at addresses after the exit point.

(<TYPE>)

When two identifier names are identical and have the same scope, you can distinguish between them by entering the type (in parentheses). Do not type a space between the identifier name and the type specification. The type will be one of the following:

filename

Specifies that the identifier is a source file.

module

These refer to module symbols. For the 68020 C compiler, these names derive from the source file name. For Ada, they are packages. Other language systems may allow user-defined module names.

procedure

Any procedure or function symbol. For languages that allow a change of scope without explicit naming, SRU assigns an identifier and tags it with type procedure.

static

Static symbols, which includes global variables. The logical address of these symbols will not change.

task

Task symbols, which are specifically defined by the processor and language system in use.

:

A colon is used to separate the HP-UX file path from the line, segment, or symbol specifier. When following the file name with a line or segment selection, there must be a space after the colon. For a symbol, there must not be a space after the colon.

Examples

The following short C code example should help illustrate how symbols are maintained by SRU and referenced in your emulation commands.

```
int *port_one;
main ()
{
int port_value;

  port_one = 255;
  port_value = 10;

  process_port (port_one, port_value);

} /* end main */
```

/users/dave/control.c

```
#include "utils.c"

process_port (int *port_num, int port_data)
{
static int i;
static int i2;

  for (i = 0; i <= 64; i++) {
    *port_num = port_data;
    delay ();
      {
      static int i;
      i = 3;
      port_data = port_data + i;
      }
    }
} /* end of process_port */
```

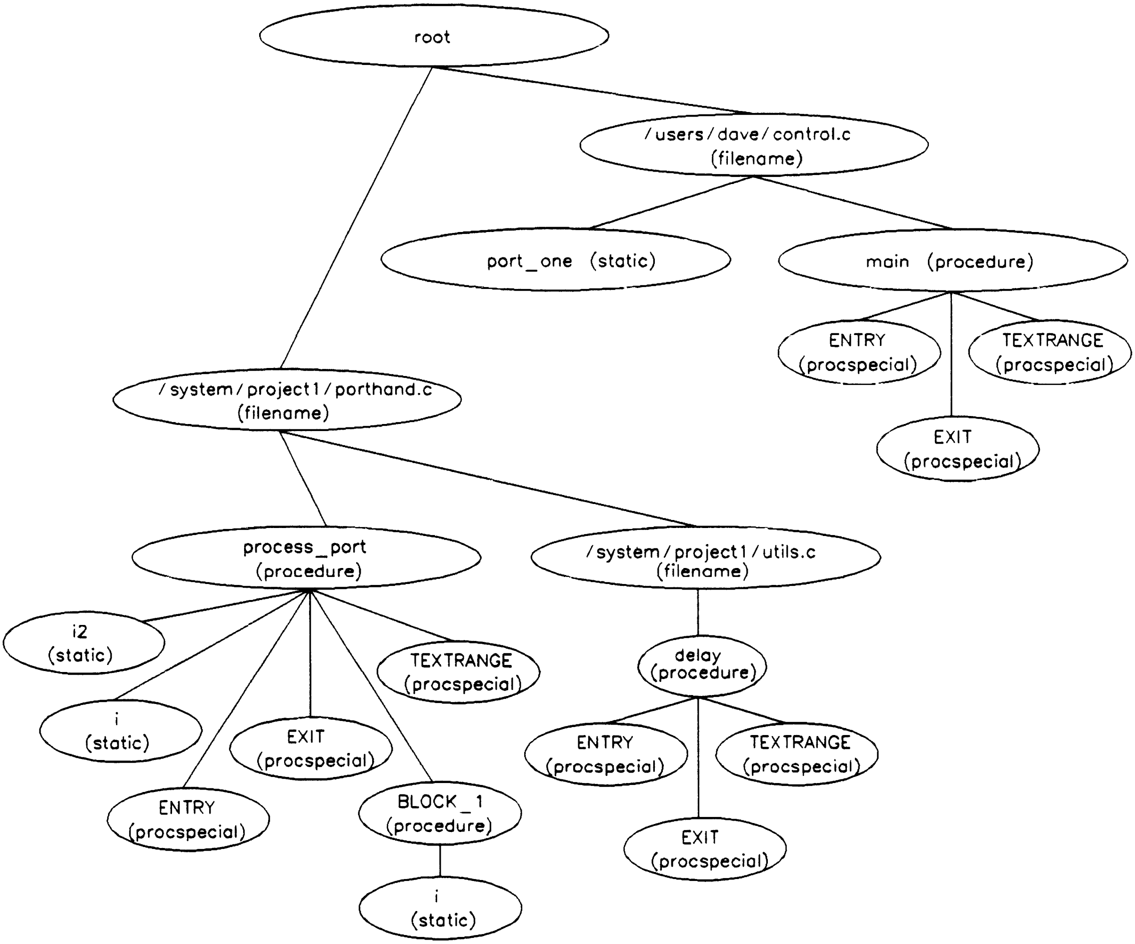
/system/project1/porthand.c

```
delay()
{
int i,j;
int waste_time;

  for (i = 0; i <= 256000; i++)
    for (j = 0; j <= 256000; j++)
      waste_time = 0;

} /* end delay */
```

/system/project1/utils.c

HP-OMF Symbol Tree

The HP-OMF (HP64000 absolute file format) symbol tree as built by SRU would appear as follows (this is not a complete symbol tree):



Note that SRU does not build tree nodes for variables that are dynamically allocated on the stack at run-time, such as i and j within the delay () procedure. SRU has no way of knowing where these variables will be at run time and therefore cannot build a corresponding symbol tree entry with run time address.

Here are some examples of referencing different symbols in the above programs:

```
control.c:main
control.c:port_one
porthand.c:utils.c:delay
```

The last example above only works with IEEE-695 object module format; the HP object module format does not support referencing of include files that generate program code.

```
porthand.c:process_port.i
porthand.c:process_port.BLOCK_1.i
```

Notice how you can reference different variables with matching identifiers by specifying the complete scope. You also can save typing by specifying a scope with cws. For example, if you are making many measurements involving symbols in the file porthand.c, you could specify:

```
cws porthand.c:process_port
```

Then:

```
i
BLOCK_1.i
```

are prefixed with porthand.c: process_port before the database lookup.

If a symbol search with the current working symbol prefix is unsuccessful, the last scope on the current working symbol is stripped. The symbol you specified is then retested with the modified current working symbol. Note that this does not change the actual current working symbol.

For example, if you set the current working symbol as

```
cws porthand.c:process_port.BLOCK_1
```

and made a reference to symbol i2, the retrieval utilities attempt to find a symbol called

```
porthand.c:process_port.BLOCK_1.i2
```

which would not be found. The symbol utilities would then strip BLOCK_1 from the current working symbol, yielding

```
porthand.c:process_port.i2
```

which is a valid symbol.

You also can specify the symbol type if conflicts arise. Although not shown in the tree, assume that a procedure called port_one is also defined in control.c. This would conflict with the identifier port_one, which declares an integer pointer. SRU can resolve the difference. You must specify:

```
control.c:port_one(static)
```

to reference the variable, and

```
control.c:port_one(procedure)
```

to reference the procedure address.

The ENTRY and EXIT symbols are accessed through the entry_exit_range keyword. For example, if you want to start execution at process_port, you can use:

```
run from porthand.c:process_port
entry_exit_range start
```

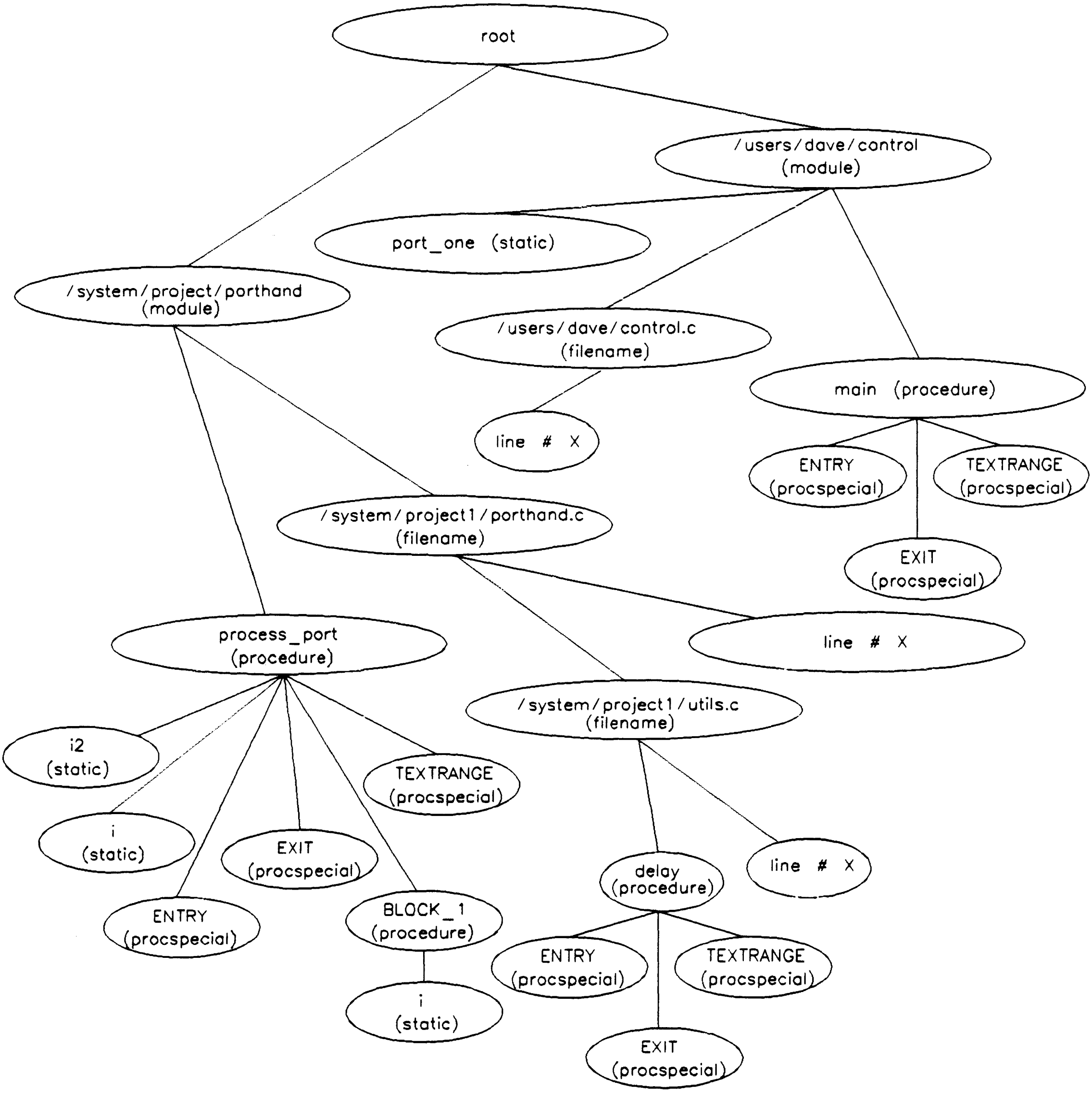
(Usually this is not necessary. SRU can generally interpret what you mean if you simply specified run from porthand.process.)

Line numbers are referenced through the source file. For example, you might want to start execution on line 5 of your source file. Type:

```
run from porthand.c: line 5
```

IEEE-695 Symbol Tree

The IEEE-695 symbol tree as built by SRU would appear as follows (this is not a complete symbol tree):



The most significant difference between this tree and the HP-OMF symbol tree is that IEEE-695 file formats use a module concept. The module owns the source file and the symbols associated with it. Compare the two tree structures to see the differences.

Now, look at the examples of referencing symbols in the previous programs. You might want to compare these to the examples given for the HP-OMF file format.

```
control.main
```

This also could be expressed as:

```
control(module).main(procedure)

control.port_one

porthand."utils.c":delay
```

You must enclose the file name in quotation marks. Otherwise, SRU interprets the period delimiting the file extension as a scope change, which is not correct.

```
porthand.process_port.i

porthand.process_port.BLOCK_1.i
```

Again, you can reference different symbols with matching identifiers by specifying the complete scope. You also can save typing by specifying a scope with cws. For example, if you are making many measurements involving symbols in the file porthand.c, you could specify:

```
cws porthand.process_port
```

Then:

```
i

BLOCK_1.i
```

are prefixed with porthand.process_port before the database lookup.

Global symbols (procedures and variables) can be specified either by their complete path through the symbol tree, or directly from the root level.

Line number symbols are owned by the source file in IEEE-695 format. To reference these, you must specify the module, then the filename, then the line number. For example:

```
porthand."porthand.c" line 5
```

You also can specify the symbol type if conflicts arise. Although not shown in the tree, assume that a procedure called port_one is also defined in control.c. This would conflict with the identifier port_one, which declares an integer pointer. SRU can resolve the difference. You must specify:

```
control.port_one(static)
```

to reference the variable, and

```
control.port_one(procedure)
```

to reference the procedure address.

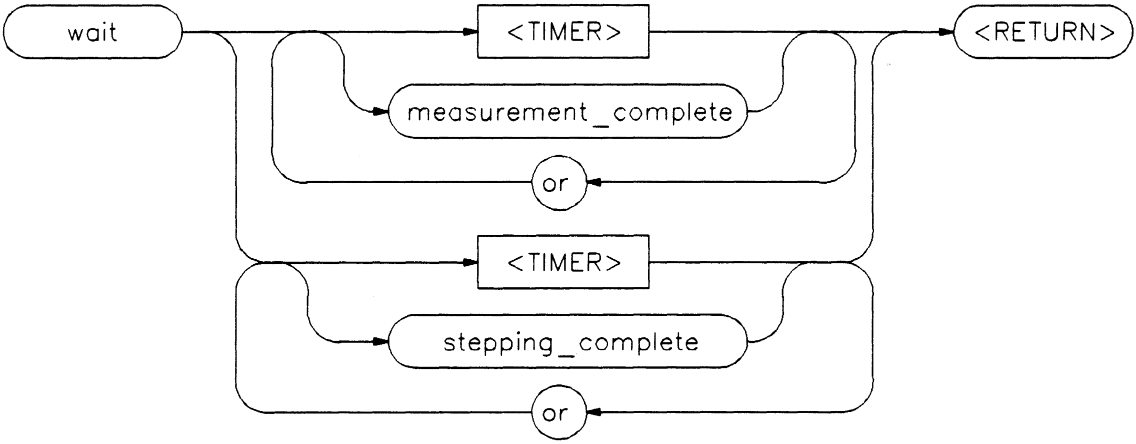
trace

Function The **trace** command allows you to trace program execution using the HP 64404 and HP 64405 Integrated Analyzers.

See the *Analysis Reference Manual for 32-Bit Microprocessors* for a detailed description of the **trace** command.

wait

Syntax



Function The **wait** command is a delay command. Delay commands are enhancements that allow flexible use of command files (although delays are also available outside command files). Command delays give the emulation system and target processor time to reach some condition or state before executing the next command. The delay commands may be included in command files.

The **wait** command is not displayed on the softkeys. You must type the command from the keyboard. After you type "wait," the wait command parameters are displayed on the softkeys.

Default Value Waiting for Ctrl C

Note

If "set intr ^ c" has not been executed on your system, replace **Ctrl c** with the **backspace** key in the following examples and parameter definitions.

Examples

`wait`	*emulator waits for* **Ctrl c** *before accepting the next command.*
`wait 6`	*emulator waits for* **Ctrl c** *or 6 seconds before accepting the next command.*
`wait measurement_complete`	*emulator waits for* **Ctrl c** *or for a pending measurement to complete. If no measurement is in progress, wait will be satisfied immediately.*
`wait measurement_complete or 20`	*emulator waits for* **Ctrl c**, *for a pending measurement to complete, or 20 seconds (whichever occurs first) before accepting the next command.*

Parameters

measurement_complete	**measurement_complete** waits for a measurement in progress to complete before the next command is executed.
stepping_complete	**stepping_complete** causes the system to wait for the current **step** command to complete before executing another command.
<TIME>	<TIME> is the number of seconds you insert for your delay.

A

User Interface Software/HP-UX Cross Reference

User Interface Command		HP-UX Command	
cat		cat	
	anychar		?
	anystrng		*
chng_dir		cd	
copy		cp	
	anychar		?
	anystrng		*
date&time		date	
edit		Defined by the variable "EDITOR"	
	recover		-r
	Readonly		-R
lifcopy		lifcp	
	binary		-b
	anychar		?
	anystrng		*
	translat		-t
	raw		-r
lifinit		lifinit	
	vol_name		-n
liflist		lifls	
	long		-l
	list_to		>
	print		|$PRINTER
lifremv		lifrm	

User Interface Command		HP-UX Command	
lifrenam		lifrename	
list_dir		ls	
	Filetype		-F
	time_mod		-t
	use_time		-u
	reverse		-r
	all		-a
	Recurse		-R
	anychar		?
	anystrng		*
	list_to		>
	print		|$PRINTER
	long		-l
log		log_commands	
	to		to
	off		off
makedir		mkdir	
manual		man	
	keyword		-k
	list_to		>
	print		|$PRINTER
move		mv	
	anychar		?
	anystrng		*
	force		-f
msconfig		msconfig	
msinit		msinit	
	search		-s
msstat		msstat	
opt_test		opt	
prom_prg		prom_prg	
removdir		rmdir	

User Interface Command		HP-UX Command	
remove		rm	
	anychar		?
	anystrng		*
	force		-f
	recurse		-r
	interact		-i
shell		!	
<system_name> (for example e386)		<system_name> (for example e386)	
tarchive		tar	
	add		r
	update		u
	extract		x
	create		c
	table		t
	anychar		?
	anystrng		*
	no_dir		o
	file/dev		f<device>
	verbose		v
	prsvmode		p
	marknow		m

Notes

Using Control Characters And Other Commands

Using Control Characters

The following control characters can be used in HP 64000-UX:

- **CTRL b** recalls commands starting from the first command you entered. You can continue pressing these keys to observe commands previously executed.

- **CTRL c** is an interrupt, and stops processing of the current command. In Option Test, this has no effect. (This is different from most HP 64000-UX interfaces, and is set this way so that the HP 64000-UX hardware is never left in an unknown state.)**

- **CTRL d** stops all tests and exits HP 64000-UX features.**

- **CTRL e** clears the command line from the cursor location to the end of the line.

- **CTRL f** rolls the diagram left while in emulation.

- **CTRL g** rolls the diagram right while in emulation.

- **CTRL l** refreshes (redraws) the display.

- **CTRL q** resumes scrolling of information on the screen (previously stopped with **CTRL s**).

- **CTRL r** recalls commands from the previous command you entered (scrolling through the commands toward the first command). You can continue pressing these keys to view previous commands.

- **CTRL s** temporarily stops scrolling of information on the screen (resume with **CTRL q**).

- **CTRL u** clears the command line.**
- **CTRL** \ (backslash) stops all tests and exits HP 64000-UX features.**
- **Tab** moves the cursor to the next word on the command line.
- **Shift Tab** moves the cursor back one word on the command line (this is for HP terminals only).

** Depends on actual stty settings.

Other Control Characters And Commands

Listed below are other control characters and commands you can use:

- **#** is used to include comments in files. All characters after the "#" are ignored when the file is executed.
- **help** or **?** displays the possible help files.
- **!** forks an HP-UX shell (using the $SHELL environment variable).
- **cd** changes directory for the present HP-UX shell.
- **<FILE> pl p2 p3** executes a command file and passes three parameters.
- **log_commands to <FILE>** puts commands you execute into a file that you specify.
- **wait** pauses a command file until you press **CTRL c** (SIGnal_INTerrupt).

- **wait measurement_complete** pauses a command file until the measurement is complete, or until **CTRL c** (SIG_INT).
- **wait <TIME>** pauses a command file until <TIME> (in number of seconds) has passed, or until **CTRL c** is pressed.

Notes

Index

HEWLETT
PACKARD

Hewlett-Packard
Printed in the USA