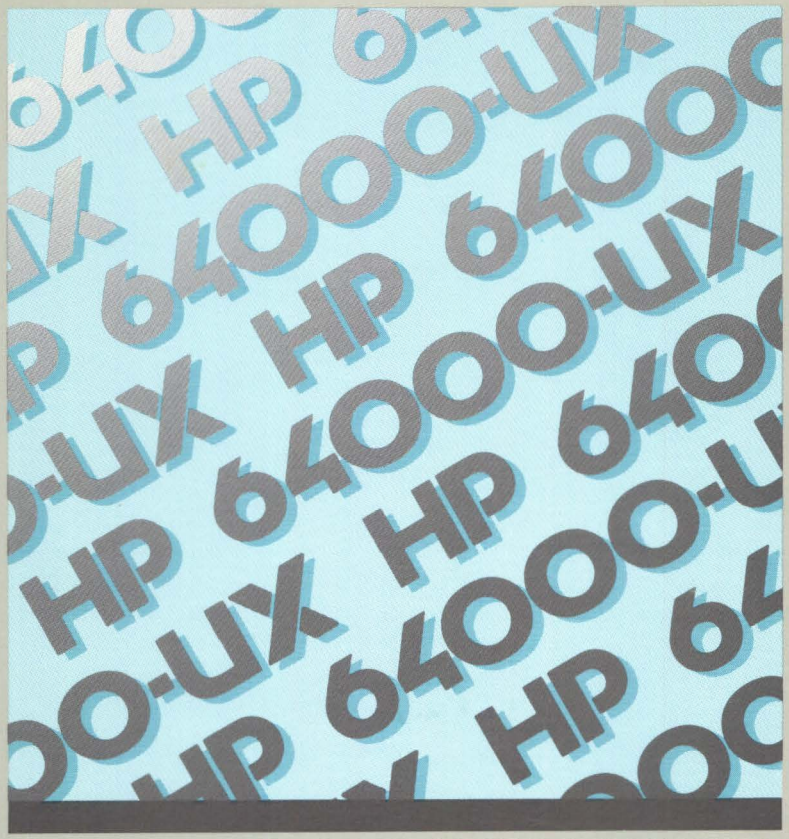
HEWLETT-PACKARD



CASE SOLUTIONS FOR MICROPROCESSORS

HP 64430

68030 Emulator Reference

DesignCenter

HP 64430

68030 Emulator

Reference Manual

HP Part No. 64430-97001
Printed in U.S.A.
February 1990

Edition 1

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**

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 or used with a three-contact to two-contact adapter with the grounding wire (green) firmly connected to an electrical ground (safety ground) at the power 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.

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 

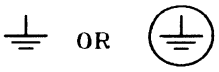
Dangerous voltages, capable of causing death, are present in this instrument. Use extreme caution when handling, testing, and adjusting.

Safety Symbols Used In Manuals

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

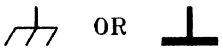
Instruction manual symbol: the product is marked with this symbol when it is necessary for the user to refer to the instruction manual in order to protect against damage to the instrument.

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 provides 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** contains brief functional and physical descriptions of the emulation system and descriptions of basic emulation features. 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** describe the use of control characters in the emulation session, and 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 is activated by executing the HP 64000-UX **pmon** command.This means that the manual will show 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 of the this manual especially useful. The cross reference table will show you how the "pmon" softkeys translate into commands that can be entered from the HP-UX prompt.

The examples provided throughout this manual use the following structure:

PRESS **edit** module.S

PRESS or press	This means you should enter a command by selecting the softkeys and/or typing in any file names or other variables which are not provided in the softkey selections.
edit	Softkeys will appear in bold type. Usually 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.
module.S	This is the name of a file which you must type in. Softkeys are not provided 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 actually executed.

Contents

2 Emulation Command Syntax

Illustrations

Tables

Notes

1

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 using an emulator have an effect on your program?
- What is happening 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 consists of several hardware modules, the emulation software, and technical manuals. The following hardware modules make up a typical 68030 emulation system:

- 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 purpose of the emulator is to aid in the development of your (target system) hardware and software design. You can use emulation during development of your system to ensure that the hardware and software being developed will work together. The emulator can be used in-circuit, alone, or with other products to debug your target system hardware and to integrate your software program modules with your target system hardware as you progress through the design phase.

Emulator Transparency

To properly perform its function, the emulator must look like the microprocessor which will eventually control your system, as seen by your target system hardware. The function, signal quality, signal timing, loading, drive capacity, and other factors at the plug-in connector should be indistinguishable from the same factors that would be present if the actual processor were being used. This characteristic is referred to as transparency.

Functional Transparency. Functional transparency refers to the ability of the emulator to function in the same way as your processor would when the emulator is connected to your target system. Total functional transparency requires that the emulator execute your program, generate outputs, and respond to inputs exactly as the actual target processor would. At the same time, the emulator must be able to give you complete and immediate 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.

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 as close as possible to the microprocessor it replaces 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.

Independent Operation

The emulation and analysis functions operate independent of the HP 64000-UX operating system. That is, once the emulation and analysis equipment has been configured and set into operation, the equipment can operate without interaction from the operating

system. This is accomplished by using a multiprocessor system for controlling the operation of the emulation system and the HP 64000-UX operating system.

Emulation Probe

The emulator allows you to replace the microprocessor in your target system with a device which performs like the microprocessor, but which 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 active probe is plugged into your target system microprocessor socket.

What Tasks Does The Emulator Do?

The tasks facilitated by an emulator are software debug, hardware debug, and hardware and software integration. These tasks are implemented by means of the following basic emulator features:

- Program Loading and Execution. Your code developed on the HP 64000-UX using the editor, compilers, assembler, and linker, or valid code developed on other systems and transferred to the HP 64000-UX host can be loaded into memory by means of the emulator and executed 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 microprocessor.
- Memory Display/Modification. You can display locations or blocks of memory and modify any memory locations that can be changed.
- Global and Local Symbols Display. You can display and find 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, and to modify them, if desired.

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

- Program Stepping. Allows you to execute code instruction-by-instruction, gaining access to the internal machine states 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.

- Breakpoint Generation. 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. Provides internal clock generation, for out-of-circuit execution.

Does The Emulation System Run Interactively With Other HP 64000-UX Modules?

The HP 64000-UX Microprocessor Development Environment allows the use of emulation and analysis features in an interactive manner between an emulator and other modules. These modules can be other emulators or analyzers. Interaction allows the integration of development work on designs, more elaborate and detailed analysis of a design, or both. The supported capabilities include:

- Simultaneous initiation of multiple measurements.
- Using the results of one measurement to control another.
- Coordinating execution of a program with the initiation of a measurement.

What Effect Does The Emulator Have On Your Program?

The effect that the emulator has on your program depends upon the emulator operating mode you select for execution. The emulator never permanently alters your program, but it may affect the execution of your program.

Real-Time Mode Vs. Nonreal-Time Mode

Depending upon the emulator operation selected for execution, 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 (except as instructed by you, and then, only for specific operations).

Interference occurs when a break to the emulation monitor is initiated either by you or automatically. The emulation monitor is a program which enables you to access the internal registers and memory of the microprocessor.

Whenever the emulator is running under control of the emulation monitor, it is no longer executing your program in real time. The emulation monitor for your emulator is described in the *68030 Emulator User's Guide.*

Real-Time Mode Capabilities

Features that typically can be performed in real-time mode are listed below.

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

Real-Time Mode Restrictions

Some features cannot be performed in real-time mode. These features require breaking 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 an access to memory which causes the emulator to break into the emulation monitor, the emulator stops executing your application code and enters the monitor. If you have circuitry that can be damaged because the emulator is not executing your application code, you should exercise special caution. You should configure the emulator to be restricted to real-time mode, and you should not break into the emulation monitor.

The features that cannot be performed in real-time mode, but require breaking into the monitor are, typically, the following:

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

The features listed above can be accessed while the emulator is configured for real-time mode by causing a break into the emulation monitor. You can cause a break when you:

- Use the break softkey.
- Cause an analysis break (for example; display registers).
- Cause a memory break (for example; attempt to access guarded memory or write to ROM).
- Cause a software break (that is; set a software breakpoint and do a run that finds the breakpoint).

What Is Happening While Your Program Is Running?

During Target Program Execution

During normal execution of your program, the emulation processor in the emulation pod generates address information for each cycle. One function of this 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, the data path buffers between your target system and the emulator processor are enabled. If the address has been mapped to emulation resource space, the data path buffers between the emulation processor and the emulation bus resources are enabled.

As your program runs, the integrated analysis circuitry observes the activity on the emulation analysis bus. Under your control, the analyzer can be instructed to store this program flow. The information can be displayed later without interrupting the real-time flow of the program.

During Emulation Monitor Program Control

The main emulation functions of the emulator are achieved by seizing control of the emulation processor from your program and transferring control to the emulation monitor so that it can extract the processor's internal information. The emulation monitor program provides the link between the emulation processor and the HP 64000-UX operating system.

The emulation monitor is actually constructed of a number of separate routines. Some of these routines are executed automatically whenever the monitor program is entered. These routines extract the internal processor information that existed at the time of entry. This information can then be displayed on the station screen for examination by the operator. If, for instance, the monitor program was entered after the execution of a program instruction, the internal machine state that existed at that time would be available.

How Does The Emulator Affect Your Microprocessor System?

The goal of the emulator is to look just like the microprocessor which will eventually control your system, as seen by your target system hardware. At the same time, it must be capable of giving you complete and immediate insight into the clock-by-clock operation of the system. The function, signal quality, signal timing, loading, drive capacity, and other factors at the plug-in pins should be indistinguishable from the same factors that would be present if the actual processor were being used. This characteristic is referred to as transparency. The *68030 Emulator User's Guide* discusses emulation functions that may affect your target system operation.

What Are The Steps To Using The Emulator?

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

- Preparing the software.
- Preparing the emulator.
- Using the emulator.

Preparing The Software

Preparing the software consists of creating and entering a program, assembling or compiling the program, and linking the assembled or compiled modules. This process is not covered in this manual. Refer to the appropriate Assembler/Linker or Compiler Manual for more information.

Preparing The Emulator

Preparing the emulator consists of properly initializing and defining a measurement system to the HP 64000-UX operating software. This task is covered in the *HP 64000-UX Measurement System Operating Manual.* After the emulator is properly defined, you configure the emulator for your particular application. Configuration is discussed in the *68030 Emulator User's Guide.*

Using The Emulator

Using the emulator consists of loading your absolute code into the emulator (provided when programs are linked), and then using the features of the emulator to observe the program as it runs, display the contents of the registers and/or memory and to debug your hardware and software. Using the emulator is covered in this manual and the *68030 Emulator User's Guide.*

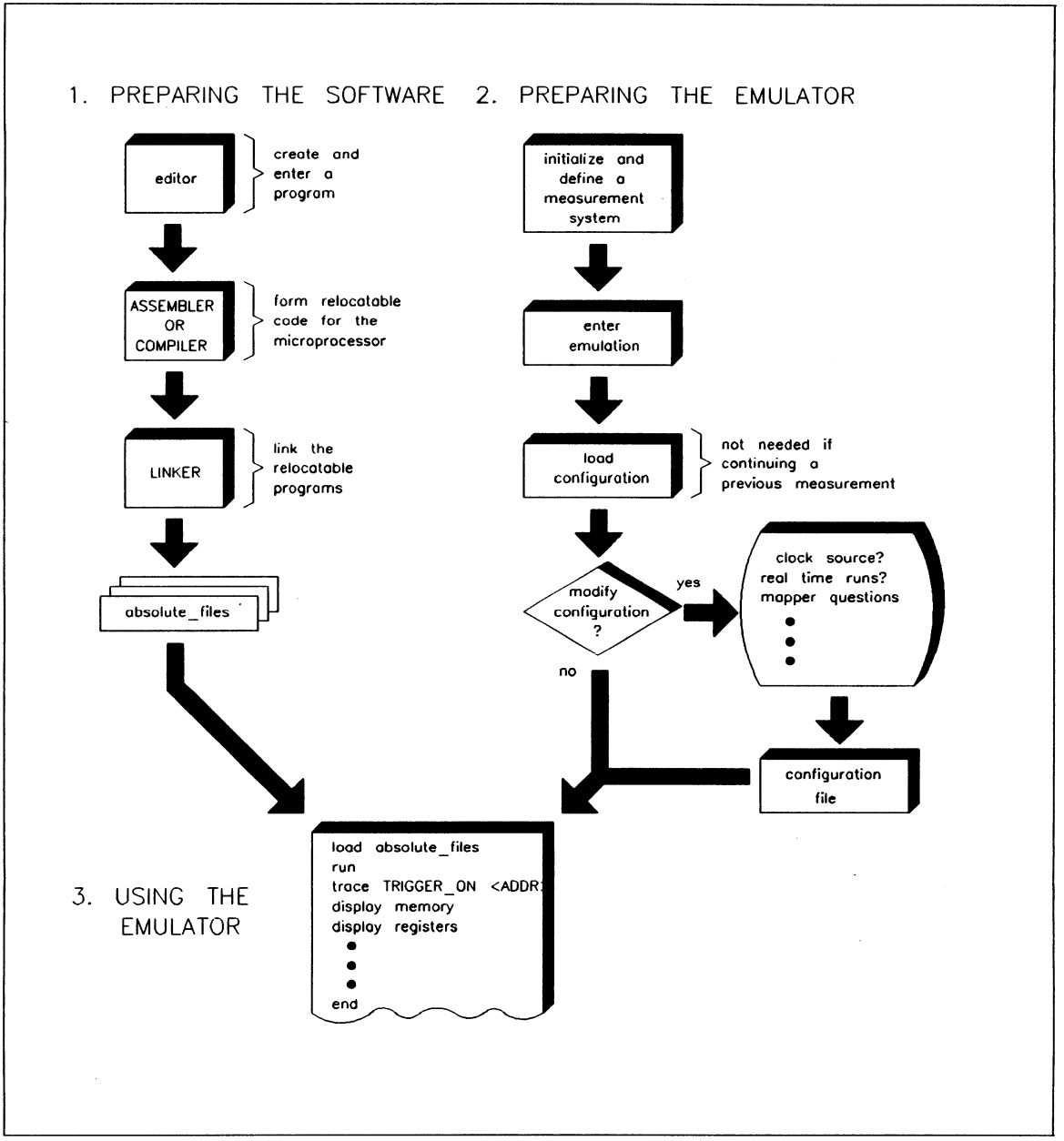


Figure 1-1 Steps to Using the Emulator

Notes

2

Emulation Command Syntax

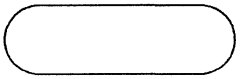
Overview

This chapter:

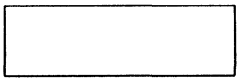
- Describes the conventions used in the syntax diagrams in this manual.
- Gives a summary of emulation commands.
- Gives a detailed description of each emulator command.

Syntax Conventions

The conventions used in the command syntax diagrams shown in this chapter are as follows:



This symbol indicates a command keyword entered by pressing a softkey. The keyword is shown as it appears in the command line and may not be the same as the softkey label.



Rectangular boxes contain either prompts indicating that parameters must be entered from the keyboard or references to additional syntax diagrams. Softkey prompts are enclosed by the "<" and ">" symbols and are shown exactly as they appear on the softkey label. **--EXPR--** and **--SYMB--** are also prompts, but allow you to access "expression help" softkeys. You can return to the normal set of emulation softkeys by pressing **--NORMAL--**. Syntax diagrams for **--EXPR--** and **--SYMB--** are included in this chapter.

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

Circles are used to 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, i.e. **copy.** Command parameters entered from the keyboard are shown in standard type.

Command Summary

A summary of emulation commands is given in table 2-1. Detailed descriptions of each command are given in the remainder of this chapter.

Table 2-1. Emulation Command List

at_execution	display local_symbols	modify analysis
break	display memory	modify configuration
copy display	display mmu_tables	modify keyboard_to_simio
copy global_symbols	display registers	modify memory
copy help	display simulated_io	modify mmu_tables
copy local_symbols	display sw_breakpoints	modify registers
copy memory	display trace*	modify sw_breakpoints
copy mmu_tables	display trace_specification*	reset
copy registers	execute	run
copy sw_breakpoints	expressions	set
copy trace*	halt	step
copy trace_specification*	help	store
display address translation	load configuration	symbol
display global_symbols	load memory	trace*
	load trace_specification	wait

* These commands are described in the *Analysis Reference Manual for 32-Bit Microprocessors*.

Note

Some command parameters shown in the following syntax diagrams may not be available when you are running emulation. What softkeys are available to you depends on how you configure the emulator for your emulation session.

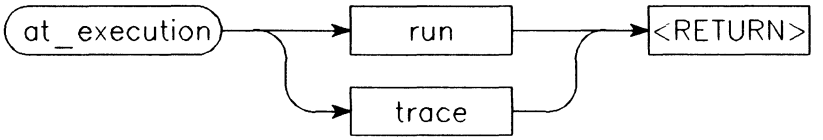
For example, if you have not configured simulated I/O to be used during your session and you enter the command:

display

the **sim_io** softkey will be not be an option on the softkey label line. Your answers to other emulation configuration questions also affect the softkey labels available to you. Only softkeys that are enabled for your emulation configuration are displayed.

at_execution

Syntax



Function

At_execution is used to prepare a run or trace command for execution. This command is used in conjunction 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 causes the run to occur. Once an execution has occurred, the run specification is removed and cannot be repeated without respecifying the run.

at_execution trace causes the trace hardware to be initialized with the given trace specification. An execute command then causes the trace to be executed. A trace specification is not removed and can be reexecuted without another at_execution trace command. **at_execution trace** and **at_execution run** can be used with a single execute command initiating both the run and the trace, and starting 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 a at_execution run command. The **at_exec**softkey label is displayed only with multiple module systems.

Default Value

none

Example

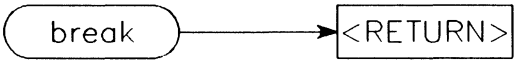
at_execution run from START
at_execution trace TRIGGER_ON a= 1234h

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** causes the processor to be diverted from execution of your program to the emulation monitor program.

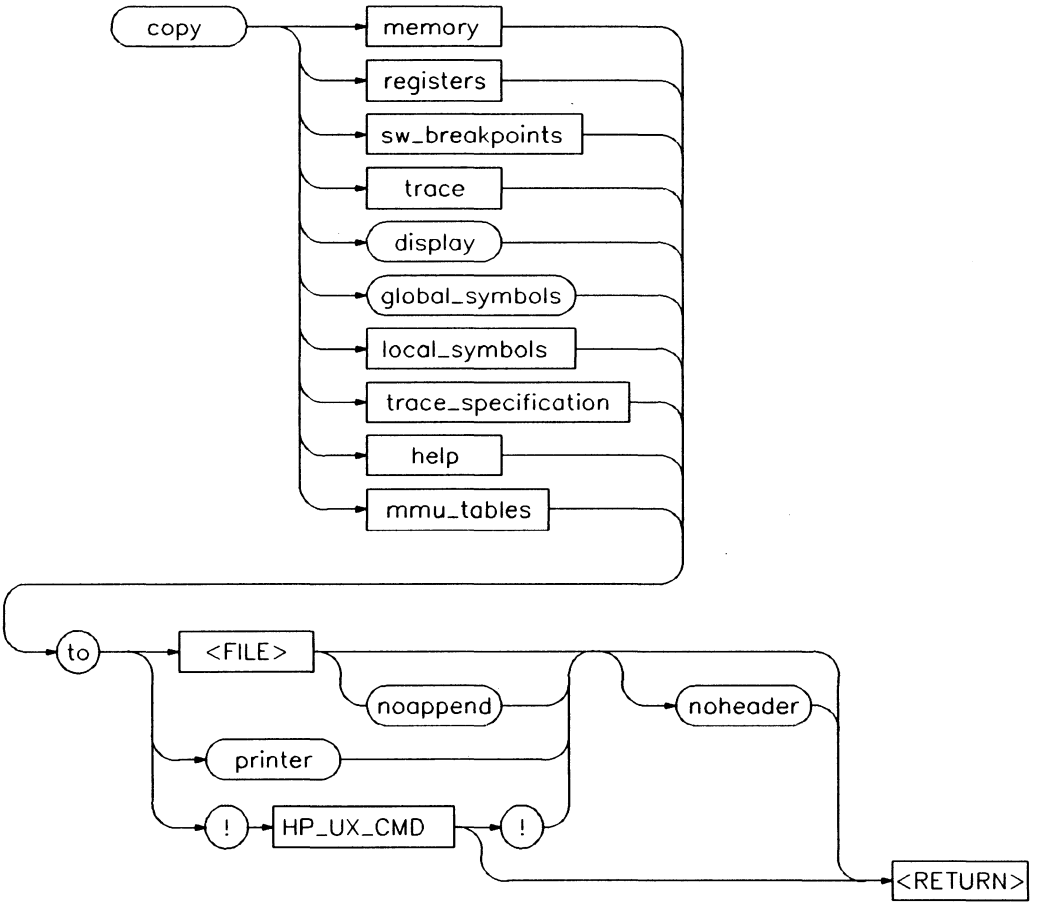
The **break** softkey is not displayed if the emulation monitor is not loaded.

Default Value none

Example **break**

copy

Syntax



Note

The mmu_tables option is not implemented in this software version.

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 what information is selected, defaults may be the options selected for the previous execution of the display command.

Parameters

display	**display** enables you to copy the information currently 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** enables you to copy a list of all global symbols in memory to the selected destination.
help	**help** enables you to copy the contents of the emulation help files to the selected destination. The keyword "help" is not available on the softkeys. It must be typed in from the keyboard. After help is typed in, the emulation help filenames are displayed on the softkeys.
HP-UX CMD	**HP-UX CMD** represents an HP-UX filter or pipe you wish to route the output of the copy command to. HP-UX commands must be preceded by an exclamation point (!). An exclamation point following the HP-UX command causes command line execution to be continued after execution of the HP-UX command. Emulation is not affected when using an HP-UX command that is a shell intrinsic.

local_symbols_in	**local_symbols_in** enables you to copy a list of local symbols in a specified source file to the selected destination.
memory	**memory** enables you to copy the contents of memory to the selected destination.
mmu_tables	**mmu_tables**, when a part of your software version, enables you to copy the MMU table information to the selected destination.
noappend	**noappend** causes the copied information to overwrite any existing file with the same name specified by <FILE>. If noappend is not specified, the default operation is to append the copied information to the end of an (existing) file.
noheader	**noheader** specifies that the information be copied without headings.
printer	**printer** specifies your system printer as the destination device for the copy command. NOTE: Before you can specify printer as the destination device, you must first define PRINTER as a shell variable. $ PRINTER=lp $ export PRINTER
registers	**registers** enables you to copy the contents of the various register sets to the selected destination.
sw_breakpoints	**software_breakpoints** enables you to copy the current software breakpoint table to the selected destination.
to	**to** enables you to specify the destination of the copied information. to must be included in the command line.

trace	**trace** enables you to copy all of, or a portion of, the current trace listing to the selected destination.
trace_specification	**trace_specification** enables you to copy all of, or a portion 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

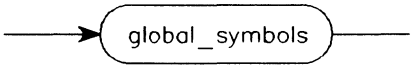
Function The copy display command copies the information currently displayed on the screen.

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 to be global (XDEF) in the source file. They include procedure names, variables, constants, and file names. When the copy global_symbols command is used, 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

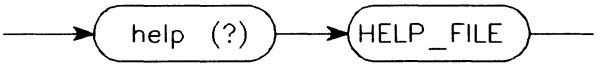
None

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. It must be typed in from the keyboard. A question mark (?) may be substituted for the keyword **help** in the command string.

Default Value

none

Examples

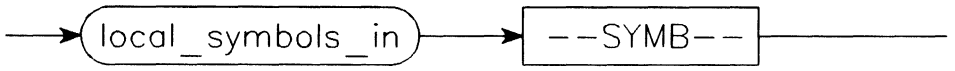
copy help **system_commands to printer**
copy ? **trace to** trc_cmd

Parameters

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

copy loc_sym

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.

Default Value none

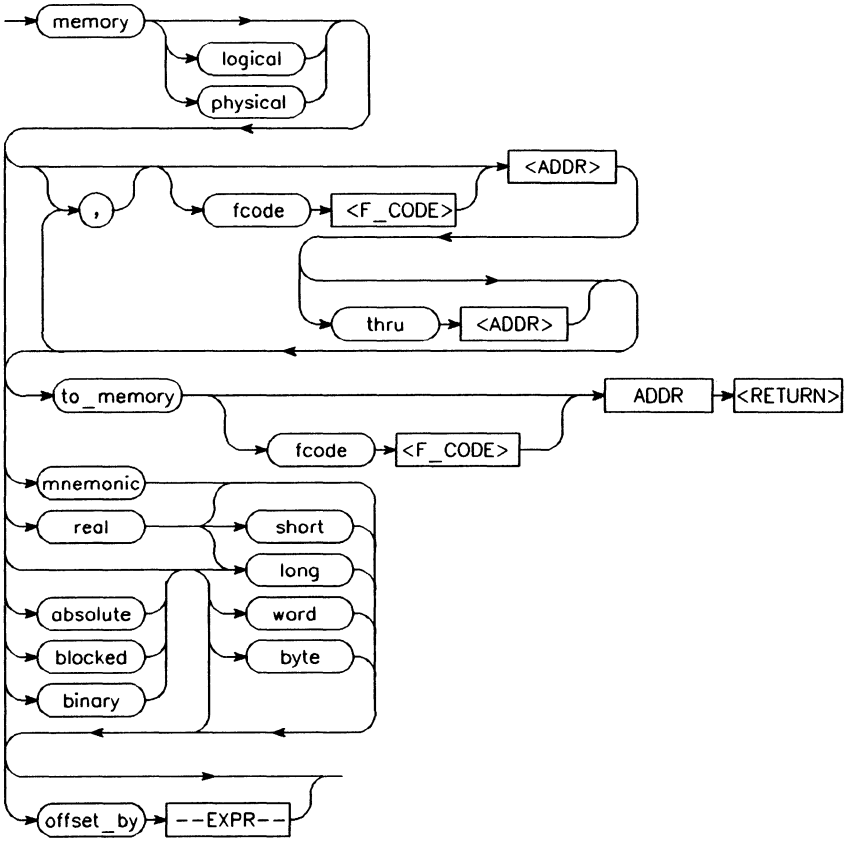
Example **copy local_symbols_in** sample.s: **to printer**

Parameters

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

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 which allows the information to be easily compared to the program assembly listing.

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** specifies that the memory listing be formatted in a single column.
<ADDR>	**<ADDR>** is a combination of numeric values, symbols, operators, and parentheses specifying a memory address or offset value. See --EXPR-- syntax diagram.
binary	**binary** specifies that the contents of memory locations be displayed as binary values.
blocked	**blocked** specifies that the memory listing be formatted in multiple columns.
fcode	**fcode** enables you to specify a function code along 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 map be specified as

a number or as a defined function code mnemonic on the softkeys.

long — **long** specifies that the memory values be copied as long word values.

When used with the **real** parameter, **long** specifies that memory be copied in a 64-bit real number format.

mnemonic — **mnemonic** causes the memory listing to be formatted in assembly language instruction mnemonics with associated operands. When specifying mnemonic format, 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** enables you to specify an offset that is subtracted from each of the actual absolute addresses before the addresses and the corresponding memory contents are listed. The value of the offset (--EXPR--) can be selected 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.

real — **real** specifies that the memory values in the listing be formatted as real numbers.

short — **short** is used with real to specify that memory values be listed as 32-bit real numbers.

thru — **thru** enables you to specify that a range of memory locations be copied.

to_memory — **to_memory** enables you to copy a block of memory to another location in memory.

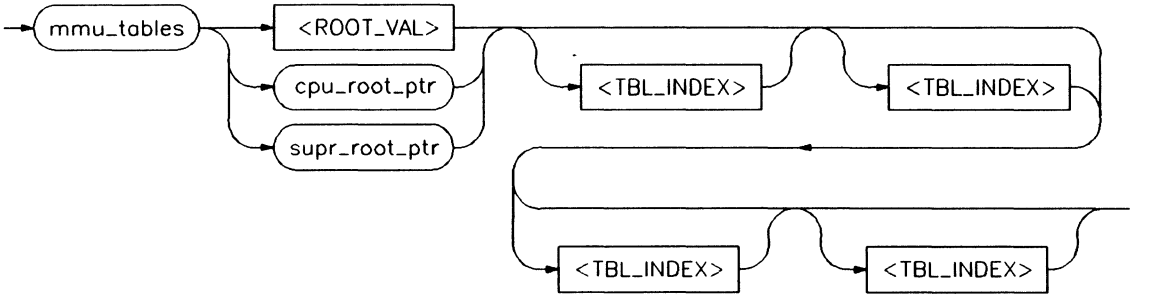
words	**words** specifies that the memory listing be copied as word values.
,	A comma (,) appearing immediately after **memory** in the command line will cause the current copy memory command to be appended to the preceding display memory command, resulting in the data specified in both commands being copied to the specified destination in the current command. The data will be 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, along with the address expression. The function code (if stated explicitly) precedes the associated address expression, and may bespecified as a number or one of the defined function code mnemonics (e.g., 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_tables

Syntax



Note

The mmu_tables option is not implemented in this software version.

Function The **copy mmu_tables** command copies the contents of the specified MMU table.

The specified MMU table can be copied to the system printer, to a listing file, or piped to an HP-UX filter.

Default Values none

Examples **copy mmu_tables cpu_root_ptr 0CH 01H**
copy mmu_tables 81808F10H 2 036H

Parameters

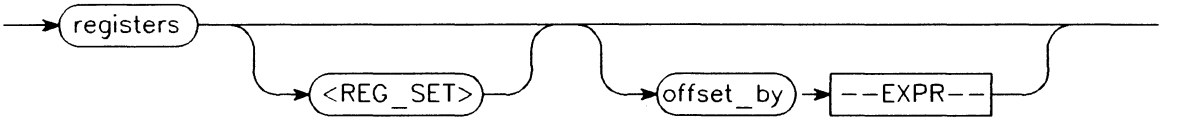
cpu_root_ptr	**cpu_root_ptr** indicates that the root pointer to be displayed is in the CPU root pointer register (CPR).
<ROOT_VAL>	**<ROOT_VAL>** is the number used to indicate the root pointer value to be used for the table access.
supr_root_ptr	**supr_root_ptr** indicates that the root pointer to be displayed is in the supervisor root pointer register (SRP).
<TBL_INDEX>	**<TBL_INDEX>** is the index into the MMU table. Each successive index offsets into the entry list pointed to by the previous indices.

Note 

Refer to 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. The emulation system must be configured for nonreal-time run mode if the registers are to be listed while the processor is running.

The listed value of the CPU program counter can be offset from the actual value by a number which allows the register information to be easily compared to the assembled listing.

When a custom coprocessor is specified, the coprocessor register set is appended to the CPU registerset listing.

Default Values

Initially cpu registers with 0 offset; thereafter last **copy registers** command specification.

Examples

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

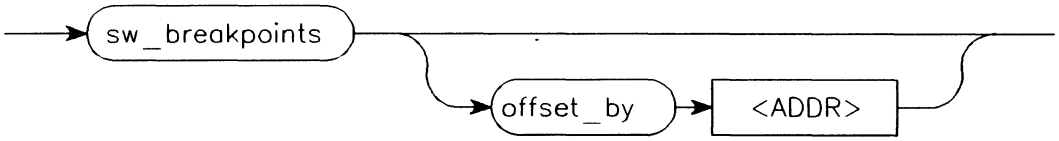
Parameters

--EXPR--	--EXPR-- is a combination of numeric values, symbols, operators, and parentheses specifying an offset value to be subtracted from the program counter. See --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 copied. The value (--EXPR--) of the offset can be selected 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. The register set names may be selected 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.

copy sw_breakpoints

Syntax



Function The copy sw_breakpoints command copies the currently defined software breakpoints and their status. If the emulation session is continued from a previous session, then the listing includes any previously defined breakpoints. The column marked status indicates whether the breakpoint is pending or inactivated. When in the pending state, a breakpoint causes the processor to enter the emulation monitor upon execution of that breakpoint. Breakpoints that have been defined as one_shot are listed as inactivated after they have been executed. Entries that show an inactive status can be reactivated by executing the modify sw_breakpoints set command.

Default Value none

Examples

copy sw_breakpoints to printer
copy sw_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 --EXPR-- syntax diagram.

offset_by

offset_by allows you to offset the listed software breakpoint address value from the breakpoint's actual address. By subtracting the offset value from the breakpoint's actual address, the system can cause the listed address to match that given in the assembler or compiler listing.

copy trace

Function The **copy trace** command enables you to copy all of, or a portion 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

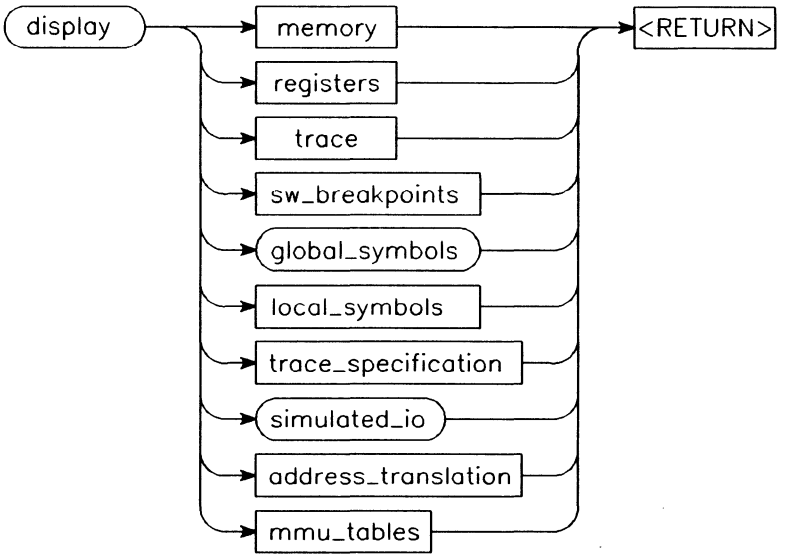
copy trace_specification

Function The **copy trace_specification** command enables you to copy all of, or a portion 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



Note

The address_translation and mmu_tables options are not implemented in this software version.

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 in some cases, the **LEFT** and **RIGHT** cursor keys to view the displayed information.

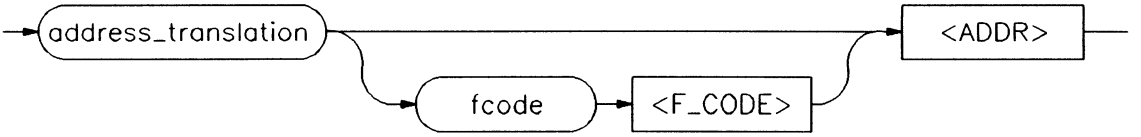
Default Values Depending on what information is selected, defaults may be the options selected for the previous execution of the display command.

Parameters

address_translation	**address_translation** allows you to display a logical to physical address translation.
global_symbols	**global_symbols** enables you to display a list of all global symbols in memory.
local_symbols_in	**local_symbols_in** enables you to display a list of local symbols in a specified source file.
memory	**memory** enables you to display the contents of memory.
mmu_tables	**mmu_tables** allows you to display mmu translation tables.
registers	**registers** enables you to display the contents of the microprocessor registers.
simulated_io	**simulated_io** enables you to display the data being written to the simio display buffer.
sw_breakpoints	**sw_breakpoints** enables you to display the current software breakpoint table.
trace	**trace** enables you to display the current trace listing.
trace_specification	**trace_specification** allows you to display your current trace specification, starting at optionally defined points.

display address_translation

Syntax



Function The display address_translation command displays the physical address that is mapped to the logical address specified.

Note

This option is not available in this software version. When it becomes available, the MMU must be correctly initialized for this feature to function.

Default Value none

Example **display address_translation fcode USER_DATA** 0F0100C00H

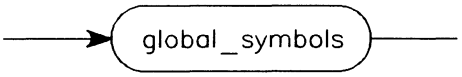
Parameters

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

fcode	**fcode** enables you to specify the function code for the address to be translated. If the MMU does not use the function codes, this value will be ignored. If the MMU does use the function codes, **cfode** can be specified to get a physical translation.
<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.

display global_symbols

Syntax



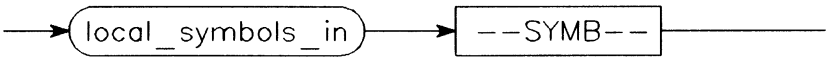
Function The display global_symbols command displays the global symbols defined for the current absolute file. Global symbols are those that are declared to be global (XDEF) in the source file. They include procedure names, variables, constants, and file names. When the display global_symbols command is used, 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

Syntax



Function The display local_symbols_in command displays the local symbols in a specified source file or scope, their addresses, their relative segment, and offset.

Default Value none

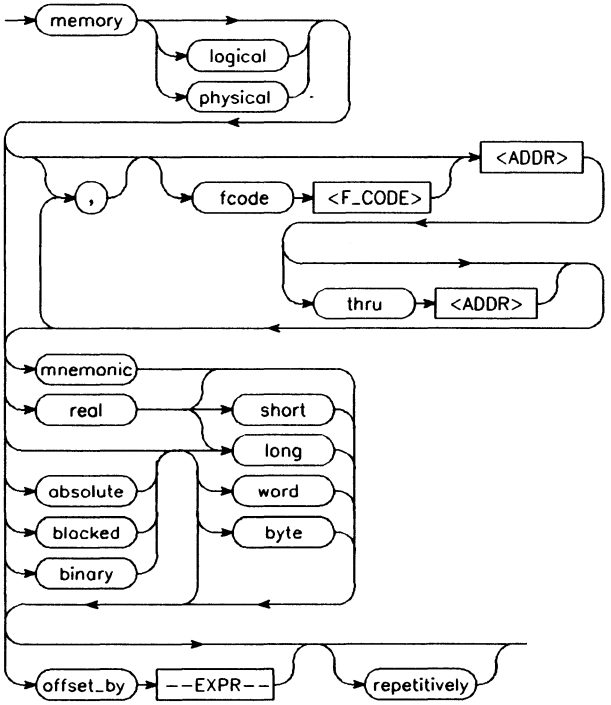
Example **display local_symbols_in** towers.c:

Parameters

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

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. In addition, the memory addresses can be listed offset by a value which 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 last one called for in a command thereafter.

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

Each of the memory access commands 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** specifies that the memory listing be formatted 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 --EXPR-- syntax diagram.
binary	**binary** specifies that the contents of memory locations be displayed as binary values.
blocked	**blocked** specifies that the memory listing be formatted in multiple columns.
fcode	**fcode** enables you to specify a function code along 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** specifies that the memory values be displayed as long word values. When used with the **real** parameter, **long** specifies that memory be displayed in a 64-bit real number format.
mnemonic	**mnemonic** causes the memory listing to be formatted in assembly language instruction mnemonics with associated operands. When specifying mnemonic format, 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** enables you to specify an offset that is subtracted from each of the actual absolute addresses before the addresses and the corresponding memory contents are listed. The value of the offset (--EXPR--) can be selected 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** specifies that the address space to be displayed is physical space.
real	**real** specifies that the memory values in the listing be formatted as real numbers.
repetitively	**repetitively** causes the system to repetitively update the memory listing displayed on your screen.
short	**short** is used with real to specify that memory values be listed as 32-bit real numbers.

thru

thru enables you to specify that a range of memory locations be displayed. Only 16 lines of information can be displayed on the screen at one time. Use the **UP** and **DOWN** cursor keys, and the **NEXT** and **PREV** keys to view additional memory locations.

words

words specifies that the memory listing be displayed as word values.

,

A comma (,) appearing immediately after memory in the command line will cause the current "display memory" command to be appended to the preceding "display memory" command, resulting in the data specified in both commands being displayed. The data will be formatted as specified in the current command.

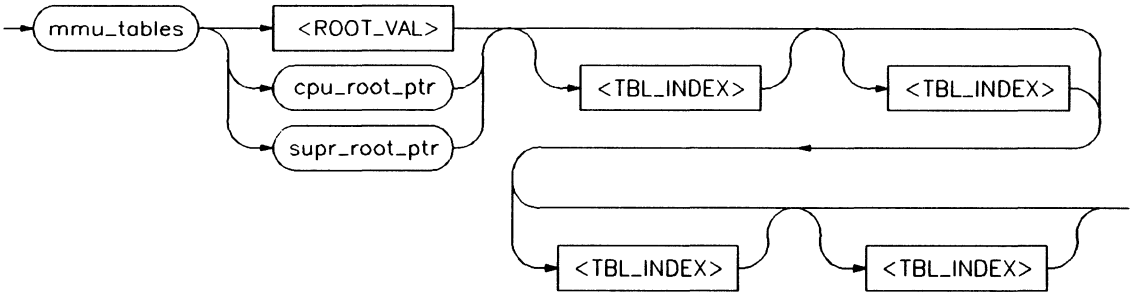
The comma is also used as a delimiter between values when specifying multiple memory addresses.

Function codes are an important part of the memory access specification, along 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 (e.g., 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_tables

Syntax



Function The **display mmu_tables** command displays the translation table entries at a given level. The tables will be indexed from the given root pointer using the index values. These values will not be checked for validity (that is, if an index goes past the end of that level, this will not be an error if the resulting value is a table entry).

Note

This option is not available in this software version. When it becomes available, this option is not present with the foreground monitor.

Default Values none

Examples **display mmu_tables cpu_root_ptr** 0CH 01H
display mmu_tables 81808F10H 2 036H

Parameters

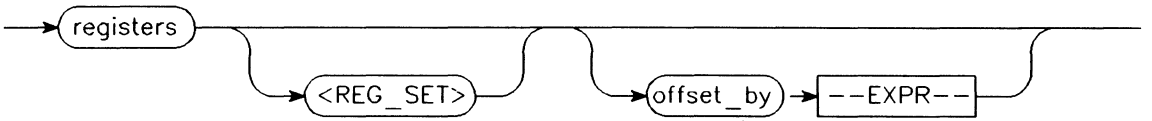
cpu_root_ptr	**cpu_root_ptr** indicates that the root pointer to be displayed is in the CPU root pointer register (CPR).
<ROOT_VAL>	**<ROOT_VAL>** is the number used to indicate the root pointer value to be used for the table access.
supr_root_ptr	**supr_root_ptr** indicates that the root pointer to be displayed is in the supervisor root pointer register (SRP).
<TBL_INDEX>	**<TBL_INDEX>** is the index into the MMU table. Each successive index offsets into the entry list pointed to by the previous indices.

Note

Refer to 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. The emulation system must be configured for nonreal-time run mode if the registers are to be displayed while the processor is running.

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

When a custom coprocessor is specified, the coprocessor register set is appended to the CPU registerset listing.

Default Values Offset is initially 0; thereafter previous value.

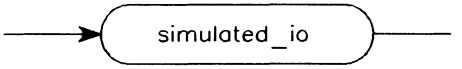
Example **display registers cpu**

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 --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. The value (--EXPR--) of the offset can be selected 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. The register set names may be selected 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.

display simulated_io

Syntax



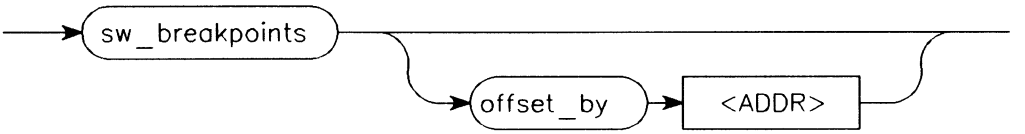
Function The display simulated_io command displays the information being written to the simulated I/O display buffer on your screen. Refer to the *HP 64000-UX Simulated I/O Reference Manual* and chapter 8 of the *68030 Emulator User's Guide* for detailed information about using simulated I/O.

Default Value none

Example **display simulated_io**

display sw_breakpoints

Syntax



Function The display sw_breakpoints command displays the currently defined software breakpoints and their status. If the emulation session is continued from a previous session, then the listing includes any previously defined breakpoints. The column marked status indicates whether the breakpoint is pending or inactivated. When in the pending state, a breakpoint causes the processor to enter the emulation monitor upon execution of that breakpoint. Breakpoints that have been defined as one_shot are listed as inactivated after they have been executed. Entries that show an inactive status can be reactivated by executing the "**modify sw_breakpoints set**" command.

Default Value none

Examples **display sw_breakpoints**
display sw_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 --EXPR-- syntax diagram.

offset_by

offset_by allows you to offset the listed software breakpoint address value from the breakpoint's actual address. By subtracting the offset value from the breakpoint's actual address, the system can cause the listed address to match that given in the assembler or compiler listing.

display trace

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

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

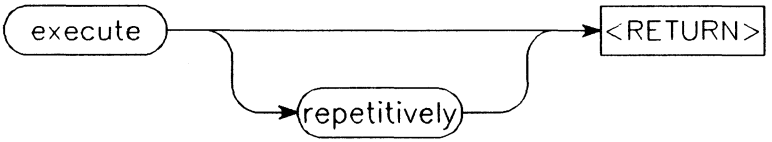
display trace_specification

Function The **display trace_specification** command enables you to display all of, or a portion of your trace specification.

See the *Analysis Reference Manual for 32-Bit Microprocessors* 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 initiated. Otherwise, the execute command is not available.

A measurement can be executed repeatedly by issuing the execute repetitively command. This restarts the current measurement after each completion, until the user issues a 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 started again by subsequent executions unless an at_execution run command is again issued.

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

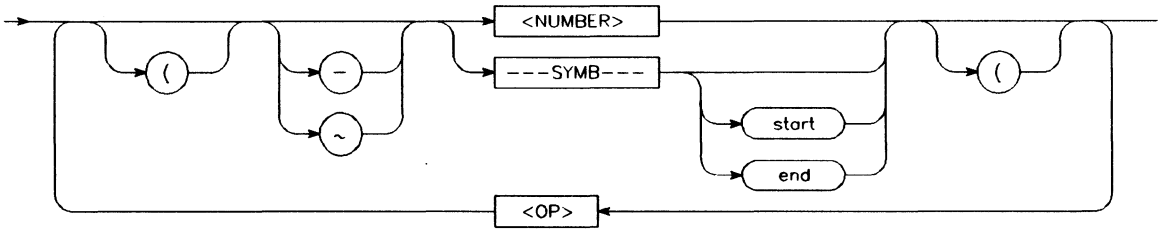
```
execute
execute repetitively
```

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.s: 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> may be (in order of precedence):
mod	(modulo)
*	(multiple)
/	(divide)
&	(logical and)
+	(plus)
-	(minus)
|	(logical or)
--SYMB--	**--SYMB--** is a symbolic reference to an address or address range, file, or other value. Symbols may be HP-UX paths, referenced line numbers in a file, file segments (prog, data, common), or global and local symbols.
start	**start** specifies that the starting address of the symbol range be used as the referenced location in the command. This parameter is useful with symbols that reference an address range rather than a single word value.
end	**end** specifies that the last address in a symbol range be used as the referenced location in the command. This parameter is useful with symbols that reference an address range rather than a single word value.
( )	Parentheses may be used in expressions. For every opening parenthesis, a closing parenthesis must exist.
-	Algebraic negation (minus)
~	logical negation (NOT)

halt

Syntax

Function The halt command stops the measurement currently executing and turns off the **repetitively** option. When the halt command is executed, some or all of the 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 causes softkey labels to be displayed, listing the areas on which you may receive help. Press the softkey for the area you are interested in, and then press the **return** key. The system will list the information to the screen using the HP-UX more utility.

The help command is not displayed on the softkeys. It must be typed in from the keyboard. A question mark (?) may be substituted for the keyword "help" in the command string.

Default Value

none

Examples

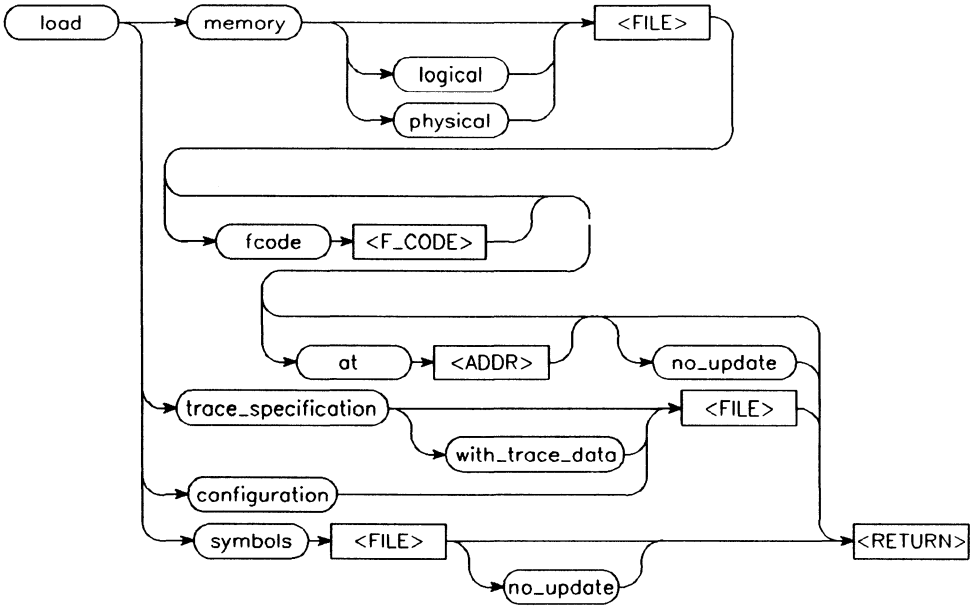
help **system_commands**
? **trace**

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 destination of the absolute code is determined by the memory configuration map which was set up during emulation configuration and the address specified during linking.

You can force the absolute code to be loaded to a location in memory 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 addess. For example, if you specify "**at**2000h", you are effectively specifying an offset of +2000h for your code.

Note

This feature should not be used if your code uses absolute addressing. Absolute addresses and symbol values in your program are not modified. This may result in run-time errors or unexpected behavior.

The load configuration command reloads an emulation configuration that you saved previously.

The load trace_specification command reloads a trace specification that you saved previously. 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 logical sort
load configuration config3
load trace_specification trace3

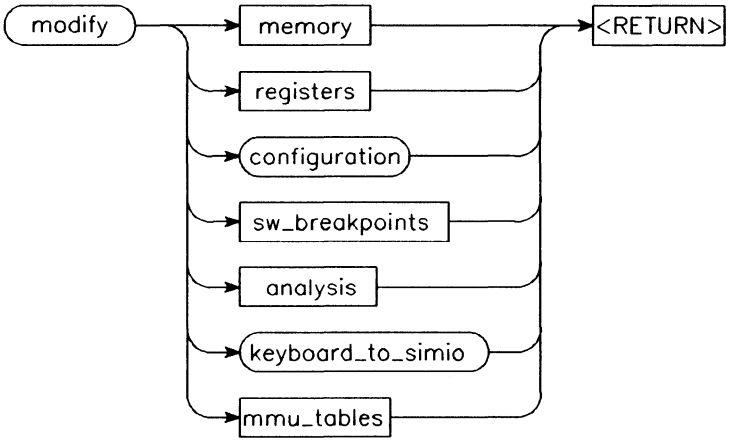
Parameters

at	**at** lets you load absolute code to a location in memory other than the address specified during linking.
configuration	**configuration** specifies that a configuration file created by a modify configuration command be loaded.
fcode	**fcode** enables you to specify a function code along 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. The function code map be specified as a number or as a defined function code mnemonic on the softkeys.
logical	**logical** specifies that the address space to be loaded is logical space.
memory	**memory** specifies that an absolute file is to be loaded into emulation or target memory.
noupdate	**noupdate** suppresses rebuilding of the symbol data base when loading an absolute file newer than its associated symbol data base. The default operation is to rebuild the symbol database.
physical	**physical** specifies that the address space to be loaded is logical space.
symbols	**symbols** specifies that the symbols for the specified file are to be loaded.
trace_specification	**trace_specification** enables you to load a specified trace file previously generated using the store trace command.
with_trace_data	**with_trace_data** specifies that the trace data be loaded along with the trace specification, if the trace data was stored.

modify

Syntax



Note

The mmu_tables option is not implemented in this software version.

Function The modify command is used to review or edit the configuration, to modify the contents of memory (as integers or as real numbers), to modify the contents of the processor registers, and to modify the analysis trace command or portions of the analysis trace specification. You can also use the modify command to modify software breakpoints.

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** enables you to modify the contents of selected memory locations.
mmu_tables	**mmu_tables**, when a part of your software version, enables you to modify the selected MMU table information.
registers	**registers** is used to modify the contents of one or more of the various register sets.
sw_breakpoints	**sw_breakpoints** sets or clears software breakkpoints used with the emulator break function.
trace_command	**trace_command** brings the last trace command back to the command line for editing.

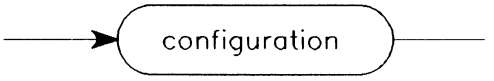
modify analysis

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



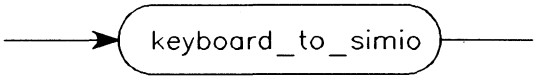
Function The modify configuration command enables you to review and edit the current emulation configuration. Each of the configuration questions is presented with the response previously entered. The prior response can be entered as displayed by pressing the **return** key, or modified as necessary and then entered by pressing the **return** key.

Default Value none

Example **modify configuration**

modify keyboard_to_simio

Syntax



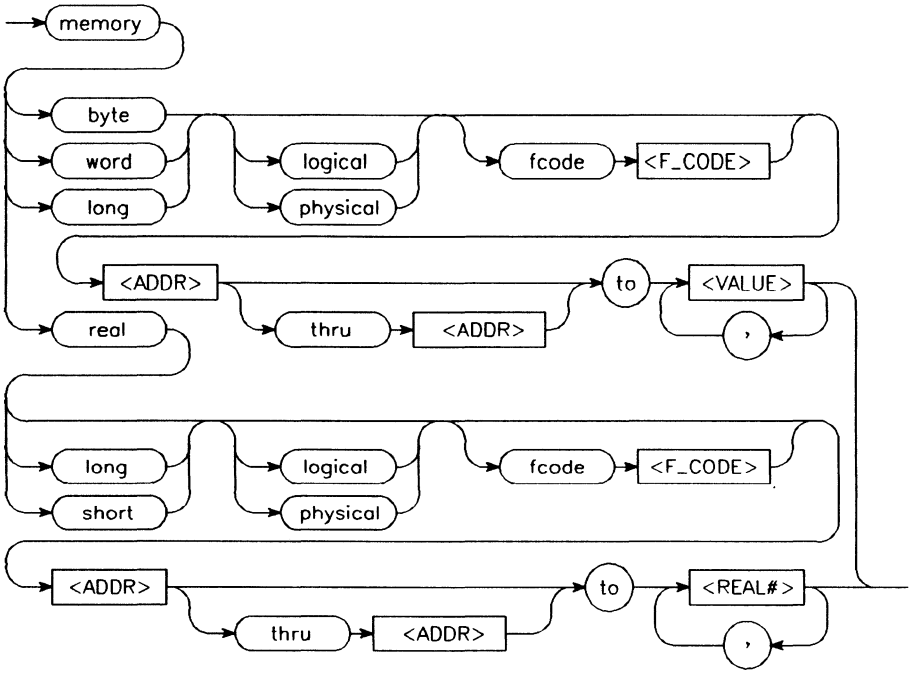
Function The modify keyboard_to_simio command activates the keyboard to interact with your program through the HP 64000-UX simulated I/O software. While the keyboard is activated for simulated I/O, its normal interaction with emulation is disabled. The emulation softkeys are blanked and the single softkey **suspend** is displayed on your screen. Pressing **suspend** and then the **return** key deactivates keyboard simulated I/O and returns the keyboard to normal emulation mode. Refer to the *HP 64000-UX Simulated I/O Reference Manual* and chapter 8 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. The command can modify the contents of each memory location in a series to an individual value or the contents of all of the 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, and may be specified as a number or one of the defined function code mnemonics (e.g. 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 is required to contain the new data, beginning with the starting address you specified.

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

Default Values

Each of the memory access commands has a separate function code default to be 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

Parameters

<ADDR>	<ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See --EXPR-- syntax diagram.
byte	**byte** specifies that the memory values be modified as byte values.
fcode	fcode enables you to specify a function code along 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 map be specified 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.
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 used as delimiters between values when modifying multiple memory addresses.

Description

A series of memory locations can be modified 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 the single address indicated is modified. When more than one value is listed, the value representations must be separated by commas.

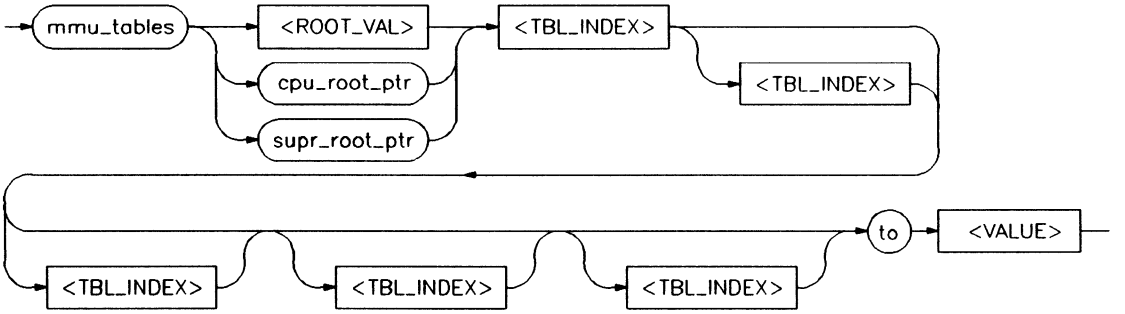
An entire block of memory can be modified such that the contents of each location in the block is changed to the single specified value, or to a single or repeated sequence. This type of memory modification is achieved by entering the limits of the memory block to be modified (--EXPR-- thru --EXPR--) 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, along 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 (e.g., 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 mmu_tables

Syntax



Note

This option is not available in this version of software. When it becomes avalable, the option is not present with the foreground monitor.

Function The **modify mmu_tables** command allows you to modify an MMU translation table entry (modify physical memory). The tables will be indexed from the given root pointer using the index values. The index values may overstep the bounds for a valid table level.

Default Values

none

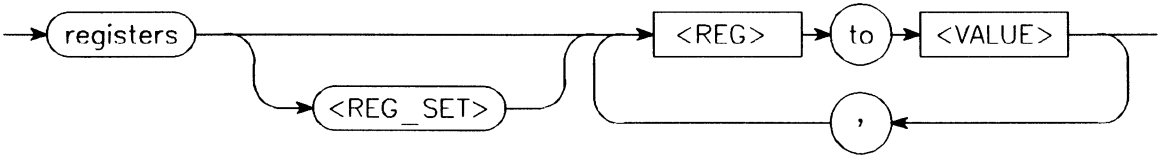
Examples **modify mmu_tables cpu_root_ptr** 4 **to** 02DF12C000H

Parameters

<ROOT_VAL>	**<ROOT_VAL>** is the number used to indicate the root pointer value to be used for the table access.
<TBL_INDEX>	**<TBL_INDEX>** is the index into the MMU table. Each successive index offsets into the entry list pointed to by the previous indices.
to	**to** enables you to specify the values to which the selected table entry will be changed.
<VALUE>	**<VALUE>** is a numeric values that you set the table entry to.

modify registers

Syntax



Function The modify register command is used to modify the contents of one or more registers in the processor/corpocessor's register set. The entry for <REG> determines which register is modified.

Register modification cannot be performed during real time running of the processor. A break must be performed to gain access to the registers.

Default Value none

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

Parameters

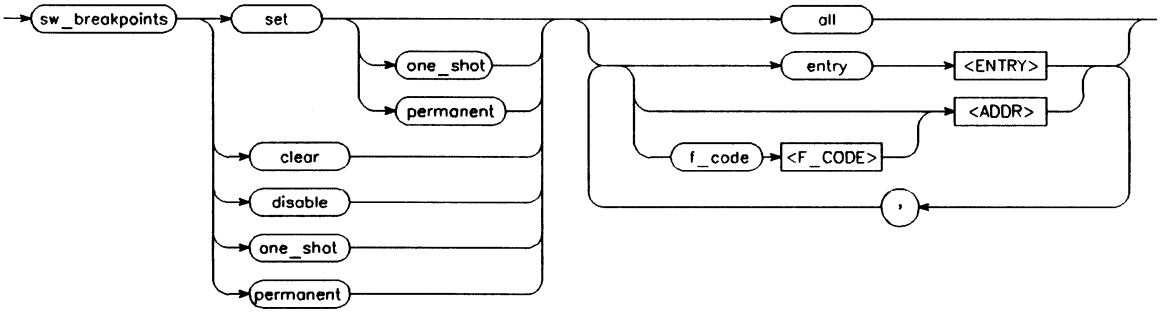
<REG> — <REG> represents the name of the register to be modified. The possible entries for <REG> are displayed on softkey labels.

<REG_SET> — <REG_SET> specifies the name of the register set to be displayed. The register set names may be selected 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.
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 an register value. See--EXPR--syntax diagram.

modify
sw_breakpoints

Syntax



Function Software breakpoints enables the emulator to "break on execution" of a specified address. Any valid address (number, label or expression) may be specified as a breakpoint. Valid addresses identify the first word of valid instructions.

Operation of the program can be resumed after the breakpoint by either a **run** or **step** command.

Default Values none

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

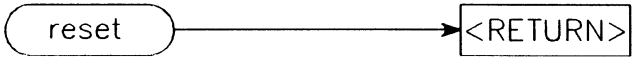
Parameters

<ADDR>	<ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a software breakpoint address. See --EXPR-- syntax diagram.
all	If used with the set parameter, **all** causes all breakpoint entries to be reactivated (set to pending). If used with the clear parameter, **all** causes all entries to be cleared and the memory locations are restored to their original values. **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 used, the function code must be specified using one of the defined function code mnemonics on the softkeys.
one_shot	**one_shot** causes the breakpoint to be set for one execution. On execution, the breakpoint is deactivated and the original contents of the memory location is restored. **one_shot** is also used to modify the mode of existing entries.
permanent	**permanent** causes the breakpoint to be set 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** enables you to set software breakpoints in your program.
,	Commas (,) are used as delimiters between specified breakpoint values.

reset

Syntax



Function The reset command suspends target system operation and reestablishes 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 question in configuration:

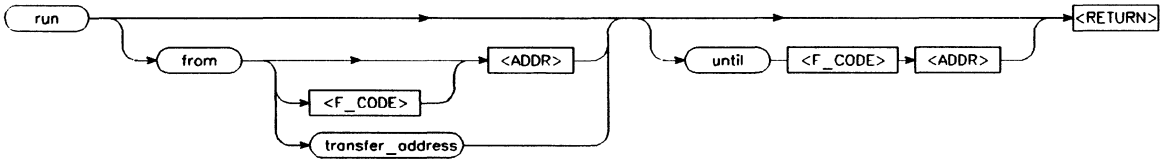
- Reset_to_monitor enabled: the processor will reset into the monitor, ignoring any user-defined reset vector.
- Reset_to_monitor disabled: the processor will vector into the reset handler defined by the user reset vector.

Default Value none

Example **reset**

run

Syntax



Function If the processor is in a reset state, run will cause the reset to be released, and if a "**from**" address is specified the processor will be directed to 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 also causes a software breakpoint to be set up at the address requested.

Default Value If the address (--EXPR--) option is omitted, the emulator will begin program execution at the current address specified by the processor's program counter.

Examples

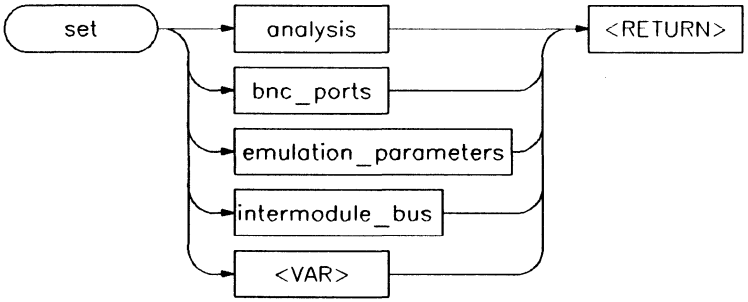
run
run from 810H
run from USER_STATE START **until** LOOP_1
run until SUPERVISOR_STATE LOOP_1

Parameters

<ADDR>	<ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See --EXPR-- syntax diagram.
<F_CODE>	<F_CODE> is a prompt for the function code. If used, the function code must be specified using one of the defined function code mnemonics on 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 The set command is used to change the configuration for analysis, the bnc ports, or the intermodule bus; to modify emulation timeout; or to set up environmental variables.

Default Value none

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 of the set analysis command.

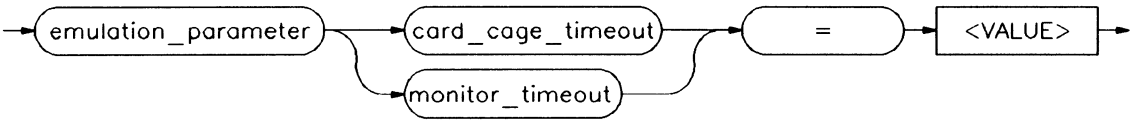
set bnc_ports

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 emulation_parameter

Syntax



Function The set emulation_parameter command lets you change either of the two emulation timeout values.

Note

The default values for card_cage_timeout and monitor_timeout will work in the vast majority of applications. These settings should be modified with great care.

Default Values 45 (card_cage_timeout, in seconds)
200 (monitor_timeout, in milliseconds)

Examples **set emulation_parameter card_cage_timeout** = 60
set emulation_parameter monitor_timeout = 800

Parameters

<VALUE>	**<VALUE>** is an integer specifying the new value for an emulation parameter. The units for <VALUE> are either seconds or milliseconds, depending on the type of parameter; seconds for card_cage_timeout, milliseconds for monitor_timeout.

card_cage_time-out	**card_cage_timeout** specifies the length of time, in seconds, that is allowed for a request to the emulator to complete. The only time this parameter should be increased is if you see the error message "HP I/O failed" on the STATUS line during normal emulation operation.
monitor_time-out	**monitor_timeout** specifies the length of time, in milliseconds, that is allowed for a request to the emulator to complete. Normally, all monitor functions (such as read/write target memory) should complete in less than 200 ms. However, if the monitor is accessing a very slow device in the target system, the value of monitor_timeout may need to be increased. The monitor_timeout value may also be set in the emulation monitor data area (refer to chapter 7).
=	Equal (=) signs are used to indicate that the emulation parameter is to be set to <VALUE>.

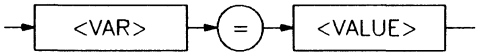
set intermodule_bus

Function The set intermodule_bus command lets you change any part of your intermodule bus (IMB) configuration.

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

set <VAR>

Syntax



Function

The set <VAR> command lets you set up "environmental" variables (aliases) for use within a particular emulation session. For example, if the following command is entered:

set x = /users/guest/test

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

load memory emulation $x/myfile

A <VALUE> that contains embedded spaces must be enclosed within quotation marks. Also, any HP-UX environmental 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 thru 10ffh"

Allowing you to use:

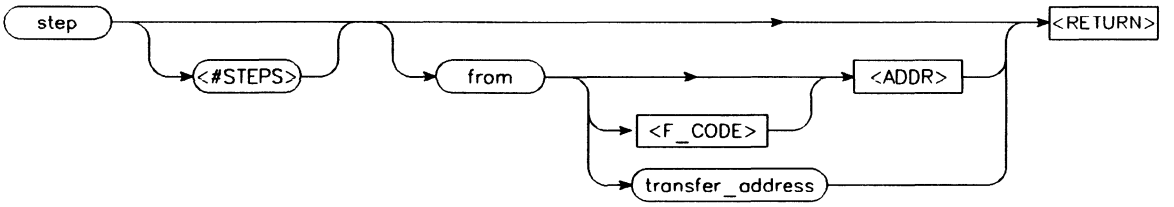
cd $emuldir
$dispmem **blocked word**

Parameters

<VAR>	**<VAR>** is a user-definable environmental variable name, consisting of a string of letters, and/or digits.
<VALUE>	**<VALUE>** is the alias assigned to the environmental variable (<VAR>), consisting of a string of letters, and/or digits.
=	Equal (=) signs are used to indicate that the environmental variable **<VAR>** is to be set to **<VALUE>**.

step

Syntax



Function The step command allows program instructions to be sequentially analyzed by causing the emulation processor to execute a specified number of instructions. 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 has been completed.

Default Values If no value is entered for <NUMBER> of times, only one instruction is executed each time the **return** key is pressed. Multiple instructions can also 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.

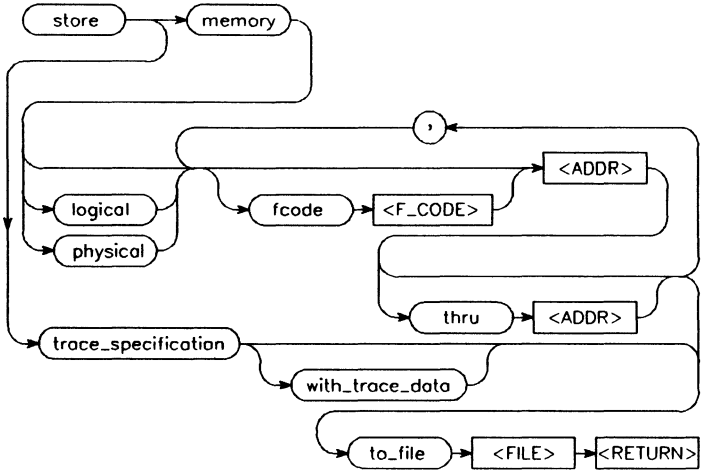
Examples **step** Return
step from fcode SUPERVISOR_STATE 810h
step 20**from fcode USER_STATE** 0A0h

Parameters

<ADDR>	<ADDR> is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See --EXPR-- syntax diagram.
<F_CODE>	<F_CODE> is a prompt for the function code. If used, the function code must be specified using one of the defined function code mnemonics on 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 The store command is used 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

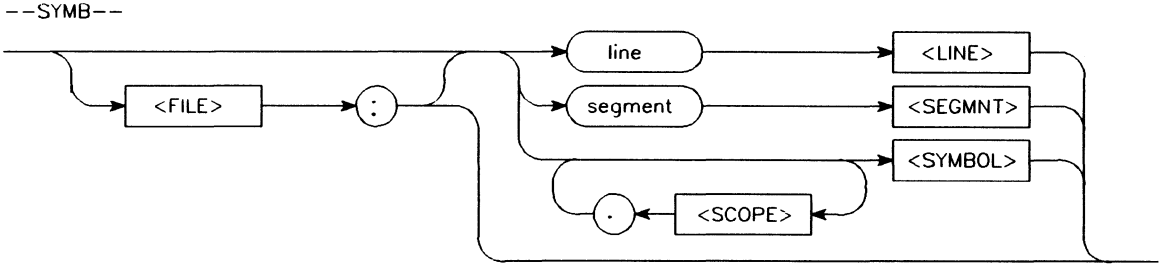
--EXPR--	--EXPR-- is a combination of numeric values, symbols, operators, and parentheses specifying a memory address. See --EXPR-- syntax diagram.

fcode	**fcode** enables you to specify a function code along 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 map 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.
logical	**logical** specifies that the selected memory locations to be stored are in logical space.
memory	**memory** specifies that the selected memory locations be stored in the specified file.
physical	**physical** specifies that the selected memory locations to be stored are in logical space.
thru	**thru** enables you to specify that memory ranges be stored.
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** specifies that the current trace specification data be stored in the specified file.
with_trace_data	**with_trace_data** specifies that the trace data be stored along with the trace specification.
,	Commas (,) are used to separate memory expressions in the command line.

Description <FILE> determines the name under which the absolute or trace file is to be stored. The store command creates a new file having the specified name as long as there is no absolute file presently on the disc with that name. In the case where a file represented by the <FILE> variable 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.

--SYMB--

Syntax



Note

If no default file has been defined by executing the **display local_symbols_in** or **load memory** commands, a source file name (<FILE>) must be specified with the first local symbol in a command line. The specified file is then used as the default file for subsequent symbols in that command line until a new source file name is specified. When the command is executed, the default file name returns to the file name specified in the last **display local_symbols_in** command (if one has been executed) or the last **load memory** command.

Function --SYMB-- is a symbolic reference to an address or address range, file, or other value. Symbols may be HP-UX paths, referenced line numbers in a file, file segments (prog, data, common), or global and local symbols.

Default Value Last file specified in a "**display local_symbols_in**" command. If **display local_symbols_in** has not been executed in the current emulation session, default is the last file specified in a **load memory** command, or none if a file has not been loaded.

Examples

```
module.S : line 5
keybd.S : scankeys.LOOP1
segment "DATA\ "
```

Parameters

<FILE>	<FILE> is an HP-UX path specifying a source file. If no file is specified, the default file is assumed, if one exists.
line	**line** specifies that the following value is a line number.
<LINE>	<LINE> prompts you to enter a line number.
<SCOPE>	<SCOPE> prompts you to enter the identifier of the portion of the program where the specified symbol is defined or active.
segment	**segment** indicates that the following string specifies a program segment (prog, data, common) in the source file.
<SEGMNT>	<SEGMNT> prompts you to enter a program segment.
<SYMBOL>	<SYMBOL> prompts you to enter a symbol name.
:	A colon (:) separates the HP-UX path specifier from the line, segment or symbol specifier. If no path specifier precedes :, then the default file is assumed for **line** or **segment**, and <SYMBOL> is assumed to be a global symbol.

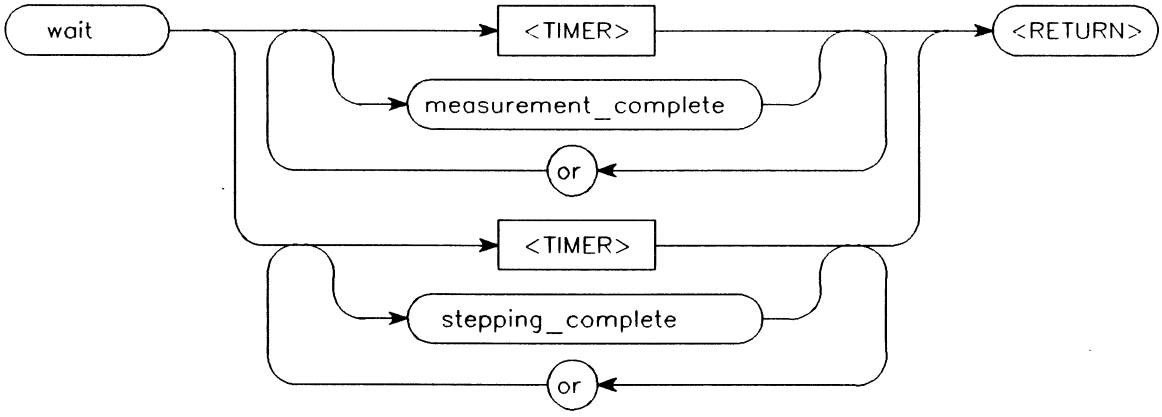
trace

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 display 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 of command files). Command delays give the emulation system and target processor time to reach some condition or state before bringing in 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 measure-ment_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 measure-ment_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

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

Notes

User Interface Software/HP-UX Cross Reference

Table A-1. User Interface/HP-UX Cross Reference

USER INTERFACE		HP-UX	
COMMAND	**OPTION**	**COMMAND**	**OPTION**
assemble		asm	
	list no_list expand no_code xref output verbose list_to print		-l -n -e -t -x -o -v > |SPRINTER
cat		cat	
	anychar anystrng		? *
chng_dir		cd	

Table A-1. User Interface/HP-UX Cross Refer. (Cont'd)

USER INTERFACE		HP-UX	
COMMAND	**OPTION**	**COMMAND**	**OPTION**
compile		comp	
	list no_list expand no_code xref output verbose list_to print		-l -n -e -t -x -o -v > | SPRINTER
copy		cp	
	anychar anystrng		? *
date&time		date	
edit		Defined by the variable "EDITOR"	
	recover Readonly		-r -R
lifcopy		lifcp	
	binary anychar anystrng translat raw		-b ? * -t -r

Table A-1. User Interface/HP-UX Cross Refer. (Cont'd)

USER INTERFACE		HP-UX	
COMMAND	OPTION	COMMAND	OPTION
lifinit		lifinit	
	vol_name		-n
liflist		lifls	
	long list_to print		-l > |SPRINTER
lifremv		lifrm	
lifrenam		lifrename	
link		lnk	
	list_to print xref output no_map no_ovlp		-l |SPRINTER -x -o -n -c

Table A-1. User Interface/HP-UX Cross Refer. (Cont'd)

USER INTERFACE		HP-UX	
COMMAND	OPTION	COMMAND	OPTION
list_dir		ls	
	Filetype time_mod use_time reverse all Recurse anychar anystrng list_to print long		-F -t -u -r -a -R ? * > |SPRINTER -l
log		log_commands	
	to off		to off
makedir		mkdir	
manual		man	
	keyword list_to print		-k > |SPRINTER
move		mv	
	anychar anystrng force		? * -f

Table A-1. User Interface/HP-UX Cross Refer. (Cont'd)

USER INTERFACE		HP-UX	
COMMAND	**OPTION**	**COMMAND**	**OPTION**
msconfig		msconfig	
msinit		msinit	
	search		-s
msstat		msstat	
opt_test		opt	
prom_prg		prom_prg	
removdir		rmdir	
remove		rm	
	anychar anystrng force recurse interact		? * -f -r -i
shell		!	
<system_name> (for example e386)		<system_name> (for example e386)	

Table A-1. User Interface/HP-UX Cross Refer. (Cont'd)

USER INTERFACE		HP-UX	
COMMAND	**OPTION**	**COMMAND**	**OPTION**
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

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 (that was 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 observe commands previously executed.

- **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 You Can Use

Other control characters and commands you can use are listed below:

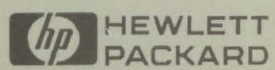
- **#** is used to include comments in files. All characters after the "#" are ignored when the file is executed.
- **help** or **?** displays the possible help files.
- **!** forks an HP-UX shell (using the SSHELL environment variable).
- **cd** changes directory for the present HP-UX shell.
- **<FILE> p1 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** (**SIG**nal_**INT**errupt).
- **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

Notes



Printed In U.S.A.