HEWLETT PACKARD

Reference

HP 64700 Emulators Terminal Interface

1

HP 64700-Series Emulators

Terminal Interface Reference

HP Part No. 64740-97003
Printed in U.S.A.
July 1989

Edition 3

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.

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, conversely, 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 November 1987 64740-90901 E1187

Edition 2 December 1988 64740-90901 E1288

Edition 3 July 1989 64740-97003

Using this Manual

This manual is a complete reference to all HP 64700 Terminal Interface commands and non-processor specific numeric and analyzer expressions.

The manual:

- Shows you the correct syntax for each command.
- Explains what the command does.
- Explains some of the side effects of a command (for example, that the emulator may break to the monitor when the command is executed).
- Shows examples of how to use the command in the context of making a measurement.
- Lists other related commands with a brief description of function.

The manual does not:

- Give you a complete tutorial on using the emulator. Refer to the *Emulator User's Guide* for your particular emulator for tutorial information.

Organization

Chapter 1 Contains all HP 64700 Terminal Interface commands in alphabetical order. Each new command starts on an odd page.

Chapter 2 Contains information on numeric and analyzer expressions which apply to all emulators. (Information on numeric expressions which are specific to certain emulators (such as address syntax) is given in the *Emulator User's Guide* for that emulator.)

Appendix A Lists the 68000 microprocessor assembly language program used in several of the command examples. A listing for a similar program for the 80186 microprocessor is also given; you can modify the examples slightly to make them work with the 80186 emulator.

Appendix B Gives a complete description of the binary/hexadecimal trace list dump format. This format allows you to write host interface programs which post process the trace list to allow you to display the trace in any style you wish. It is the only way that external timing analyzer information may be recovered from the trace.

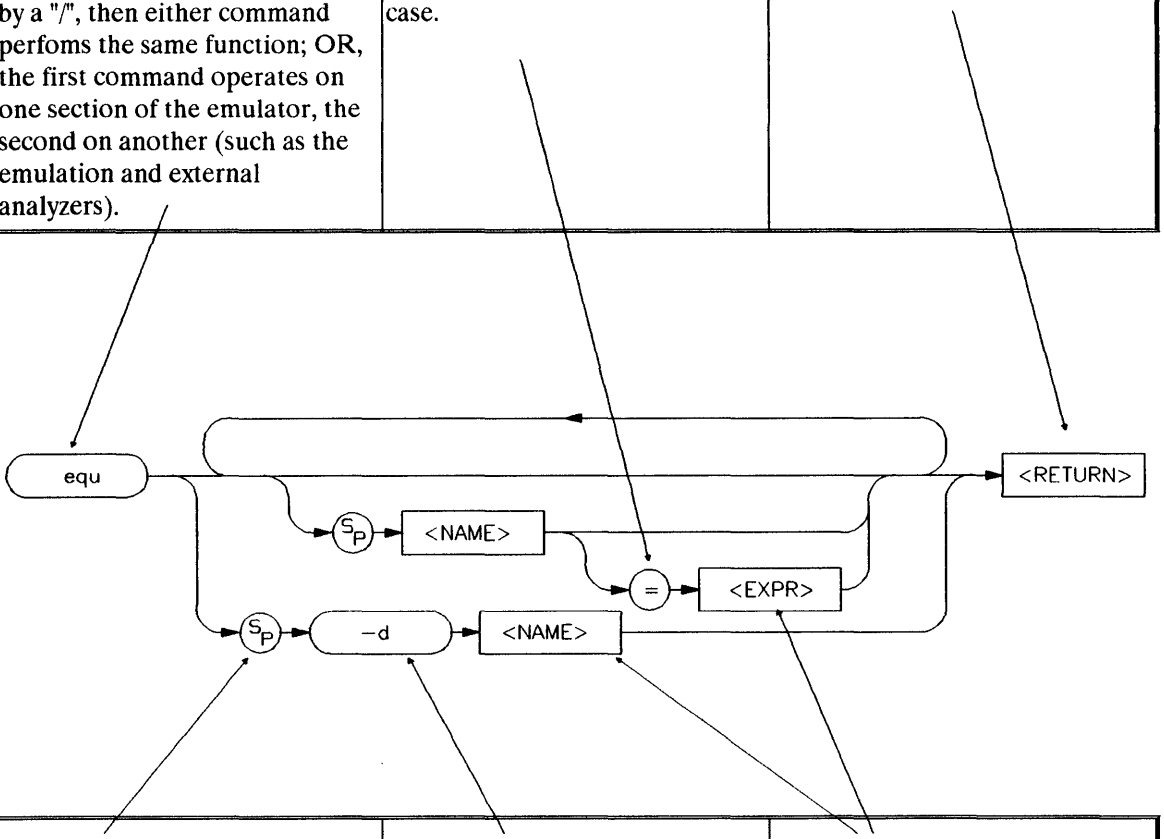
Appendix C Contains descriptions of error messages that can occur while using the Terminal Interface. The error messages are listed in numerical order, and each description includes the cause of the error and the action you should take to remedy the situation. Error messages described in this appendix are "generic"; that is, they can occur in any of the HP 64700 emulators. Errors specific to a particular emulator are described in the *Emulator User's Guide.*

Appendix D Contains information on entering commands. Included are descriptions of emulator prompts, command recall and abort, multiple commands on the same line, and comments in commands.

Manual Conventions

Syntax diagrams used in this manual are interpreted as shown in the following diagram.

All command syntax diagrams start with the command name. If there are two names, separated by a "/", then either command perfoms the same function; OR, the first command operates on one section of the emulator, the second on another (such as the emulation and external analyzers).	Operators such as "=" appear either in circles or bubbles; they have the same function in either case.	All command lines must be terminated by a carriage return.



This indicates a required space in all of the command diagrams. You should only insert the number of spaces shown on the diagram, as some commands are sensitive to additional spacing.	Command options are shown in bubbles such as this one. (Some operators are also depicted in bubbles.)	All identifiers in rectangles (except for <RETURN>) indicate syntatic items which are further defined, either in the command description text, or by another set of syntax pages. (For example, <NAME> is completely defined under the **equ** command; <EXPR> has its own set of descriptive pages.)

Are You Following the Map?

Before using this manual, you should make sure that you have followed the steps presented in the *HP 64700 Emulator Documentation Map* (the Read Me First document). If you have not, please retrace your steps and read the system overview manual, install your hardware, then work through the tutorials in the *Emulator User's Guide* before using this manual.

Contents

1

Emulator Commands

This chapter consists of HP 64700-Series Emulator/Analyzer Terminal Interface commands and descriptions. The syntax, functional description, parameters, default values, examples of command usage, and related commands are included.

Summary of Commands

COMMAND	DESCRIPTION
b	Break the emulator to monitor
bc	Specify break conditions
bnct	Specify BNC signal drivers and receivers
bp	Insert or modify software breakpoints
cf	Set emulator specific configuration items
cim	Copy target memory to emulation memory
cl	Control command line editing
cmb	Enable/disable CMB interaction
cmbt	Specify drivers and receivers of CMB trigger

cov	Measure percentage of memory locations accessed
cp	Copy memory blocks
dt	Set or display system date/time
dump	Dump memory to a host file
echo	Echo character strings or expressions
equ	Equate names to expressions
es	Display emulator status
help,?	Display help information for commands
init	Initialize the emulator
lcd	Load a block of emulator system control code
load	Load user programs into emulation or target memory
m	Display/modify memory locations
mac	Define command macros
map	Map emulation and target system memory
mo	Set global memory access and display modes
po	Assign ports, redefine prompt, dump command files
pv	Run emulator/analyzer performance verification
r	Run the emulator from current PC or specified location

reg	Display/modify processor registers
rep	Repeat a group of HP 64700 commands
rst	Reset the emulation microprocessor
rx	Specify starting address for coordinated emulator run
s	Step the emulation processor
ser	Search emulation or target memory for values
stty	Set data communications parameters
sym	Manage the emulator symbol table
t,xt	Start an analyzer trace
ta	Display analyzer line activity
tarm,xtarm	Specify arming condition for analyzers
tcf,xtcf	Set analyzer configuration to easy or complex
tck,xtck	Specify analyzer master clock qualifiers
tcq,xtcq	Specify analyzer trace tag count qualifier
telif,xtelif	Specify sequencer secondary branch qualifier
tf,xtf	Specify the trace list display format
tg,xtg	Specify a trigger condition for the analyzer
tgout,xtgout	Specify signals to drive upon analyzer trigger
th,xth	Halt the analyzer
tif,xtif	Specify sequencer primary branch qualifiers

tinit	Reset trace specification
tl,xtl	Display/dump current trace list
tlb,xtlb	Define labels for analyzer input lines
tp,xtp	Specifies location of trigger state in trace list
tpat,xtpat	Specify analyzer complex configuration patterns
tpq,xtpq	Specify trace prestore qualifier
trng,xtrng	Specify a complex configuration range qualifier
ts,xts	Display status of analysis trace
tsck,xtsck	Specify analyzer slave clocks
tsq,xtsq	Manipulate the trace sequencer
tsto,xtsto	Specify analyzer trace storage qualifiers
tx,xtx	Set analyzer to trace on receipt of CMB /EXECUTE
ver	Display Terminal Interface firmware version number
w	Wait for specified event
x	Start synchronous CMB execution
xp	Enable/disable transparent mode
xteq	Specify external timing analyzer edge trigger
xtgq	Specify external timing analyzer glitch trigger
xtm	Specify external timing analyzer mode

xtmo	Specify external analyzer mode
xtsp	Define external timing analyzer sample period
xtt	Specify external timing analyzer trigger condition
xttd	Specify external timing analyzer trigger delay
xttq	Specify external timing analyzer transition trigger
xtv	Set threshold voltages for external analyzer probes

Notes

b

Summary

Break the emulator to monitor

Syntax

Function

The **b** command issues a break to the emulator, causing it to stop executing the user program and begin execution of the monitor program. If the emulator is in the reset state when a break occurs, it will be released from reset and will begin execution within the emulation monitor.

Parameters

None.

Defaults

None.

Examples

To break the emulation microprocessor into the monitor, type:

```
U> b
```

You will see:

```
M>
```

Related Commands

r (runs the user program from the current pc or a specified address)

s (steps the user program a number of instructions from the current pc or a specified address)

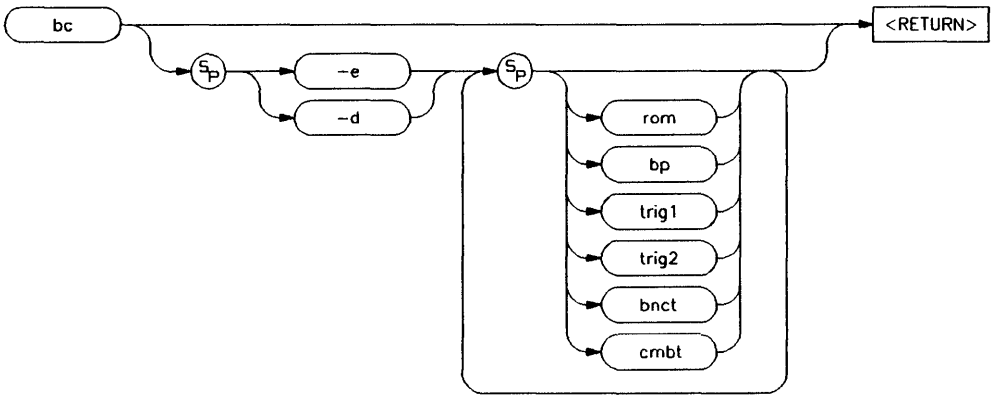
Notes

bc

Summary

Specify break conditions

Syntax



Function

The **bc** command allows you to set break conditions for the emulation system. You can independently enable or disable six different break conditions: write to ROM, software breakpoints, breaks due to assertion of the BNC or CMB trigger signals, and breaks due to the assertion of the internal **trig1** and **trig2** signals. This allows you to have the emulator break to the monitor upon error conditions (such as write to ROM or finding a software breakpoint in a piece of code it never should have reached), or break to the monitor when an analyzer measurement has completed.

When you use the **bc** command, the emulator may break into the monitor while each enable/disable is being executed. If the emulator was executing your program when the **bc** command was received, it will return to your program when finished executing the command. If you request only a display of the current break conditions, the emulator does not break to the monitor.

A hardware reset which occurs during processing of the **bc** command may result in the particular break condition being left in an unknown state. If this occurs, a display of the break conditions will show a question mark "?" instead of **-e** or **-d** next to the break condition.

Parameters

-e	Enables the indicated break conditions (which must be specified immediately following the **-e** on the command line).
-d	Disables the indicated break conditions (which must be specified immediately following the **-d** on the command line).

Note 

The options **-e** and **-d** cannot both be specified within the same **bc** command.

---

rom	Enable/disable emulator breaks to monitor on occurrence of a write to ROM by the user program.

Note 

A microprocessor like the 68000 with a pipeline architecture begins execution of the next instruction before it completes execution of the current instruction. Since a write to ROM cannot be detected until the bus cycle which causes it completes, situations will arise where instructions after the instruction that caused the write to ROM to occur will execute.

---

bp	Enable/disable recognition of software breakpoints inserted with the **bp** command.

Note

The "breakpoints" break condition should not be disabled (**bc -d bp**) while the emulator is running user code. If this command is entered while the emulator is running user code, and the emulator is executing code in the area where the breakpoints are being modified, program execution may be unreliable. (Breakpoints are modified as a result of the **bc -d bp** command because enabled breakpoints are replaced by the original opcodes when the "breakpoints" break condition is disabled.)

bnct	Enable/disable breaks generated by assertion of the **bnct** (rear panel BNC) signal. Note that this signal may also drive either the **trig1** or **trig2** signals; or, it may drive both.
cmbt	Enable/disable breaks upon assertion of the CMB (Coordinated Measurement Bus) trigger signal. Note that the CMB trigger signal may also drive either the **trig1** or **trig2** signals; or, it may drive neither or both.
trig1	Enable/disable breaks generated by assertion of the **trig1** (trace trigger one) signal. Refer to the **tgout**, **bnct**, and **cmbt** commands for information on specifying drivers and receivers of the **trig1** signal.
trig2	Enable/disable breaks generated by assertion of the **trig2** (trace trigger two) signal. Refer to the **tgout**, **bnct**, and **cmbt** commands for information on specifying drivers and receivers of the **trig2** signal.

Defaults

If no parameters are specified, the enable/disable status of all six break conditions is displayed.

Examples

To display the status of all six break conditions, type:

```
M> bc
```

You will see something similar to the following:

```
bc -e bp #enable
bc -d rom #disable
bc -d bnct #disable
bc -d cmbt #disable
bc -d trig1 #disable
bc -d trig2 #disable
```

To enable breaks on write to ROM and upon assertion of the **trig1** signal, and disable software breakpoints and breaks generated by the **trig2** signal, type:

```
M> bc -e rom trig1
M> bc -d bp trig2
```

Related Commands

bnct (specify drivers and receivers of the rear panel BNC signal)

cmbt (specify drivers and receivers of the CMB trigger signal)

bp (set/delete software breakpoints)

map (specify whether memory locations are mapped as RAM or ROM)

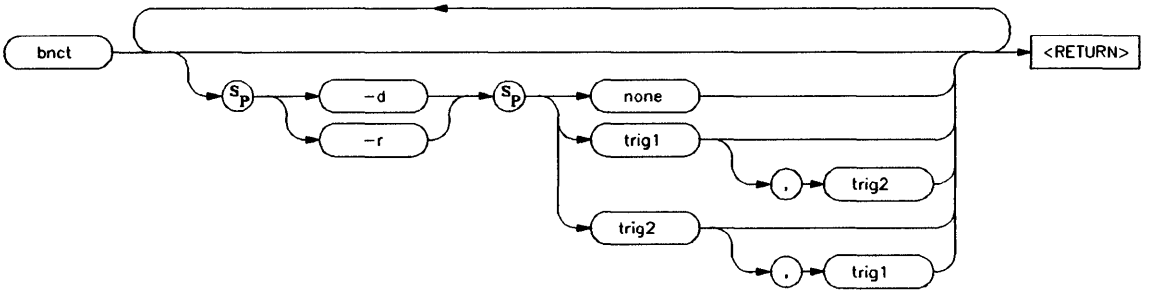
tgout (specify whether the **trig1** and/or **trig2** signals are to be driven when the analyzer finds the trigger condition)

bnct

Summary

Specify BNC signal drivers and receivers

Syntax



Function

The **bnct** command allows you to specify which of the internal **trig1/trig2** trigger signals will drive and/or receive the rear panel BNC trigger. You can specify the signals individually, as an ORed condition for drive, or as an ANDed condition for receive; or, you can specify that the signals are not to be driven or received.

Normally, you would use this command to cross-trigger instruments. For example, you may wish to trigger a digitizing oscilloscope hooked to various timing signals when the emulation analyzer finds a certain state, or, you may wish to do the converse and trigger the HP 64700's analyzer when an oscilloscope finds its trigger.

Parameters

-d — The **-d** parameter indicates that the BNC port will drive the triggers, trig1 and trig2, to the emulator's internal analyzer.

-r	The **-r** parameter causes the BNC port to receive the triggers, trig1 and trig2, from the analyzer, and send them out the BNC port.
none	If you specify **none** with the **-d** option, then the rear panel BNC signal will not drive either of the analyzer triggers. If you specify **none** with the **-r** option, the rear panel BNC will not receive trig1 or trig2 from the internal analyzer.
trig1	If **trig1** is specified, then the internal "trig1" signal will drive or receive the BNC signal, depending on whether you specified the **-d** or **-r** option.
trig2	If you specify **trig2**, then the internal "trig2" signal will drive or receive the BNC signal, depending on whether you specified the **-d** or **-r** option.

Note 

You can also specify that both the **trig1** and **trig2** signals are to drive or receive the BNC signal. To do this, place a comma between the two signals on the command line.

Defaults

If no options are specified, the current setting of **bnct** is displayed. Upon powerup, **bnct** is set to **bnct -d none -r none**.

If you specify one of the **-d** or **-r** options without the other, the other option is left in the same state it was in before the command was entered.

Examples

To view the current **bnct** setting, type:

```
M> bnct
```

You will see:

```
bnct -d none -r none
```

If you want to trigger an instrument hooked to the BNC when the HP 64700 analyzer finds its trigger, you might do the following:

```
M> tcf -e
M> tg addr=2000
M> tgout trig1
M> bnct -d none -r trig1
```

By specifying this command sequence, the external instrument will be triggered when the emulation processor reaches the trigger pattern of address=2000.

The reverse situation is where you want to trigger the HP 64700 analyzer when an external instrument finds its trigger. Type:

```
M> bnct -d trig1 -r none
M> tarm =trig1
M> tg arm
```

Note

You should not set up an analyzer in an emulator to both drive and receive the same trigger signal. For example, if you issued the commands **tg arm**, **tarm =trig1**, **tgout trig1**, and **bnct -d trig1 -r trig1**, then the analyzer **trig1** signal will become latched in a feedback loop and will remain latched until the loop is broken. To break the loop, you must first disable the signal's source, then momentarily disable either the drive or receive function. In this case, the commands **tgout none** and **bnct -d none** will break the loop.

Related Commands

bc (break conditions; can be used to specify that the emulator will break into the emulation monitor upon receipt of one of the **trig1**/**trig2** signals)

cmbt (coordinated measurement bus trigger; used to specify which internal signals will be driven or received by the HP 64700 coordinated measurement bus)

tarm (analyzer trace arm; used to specify arming (begin to search for trigger) conditions for the analyzer -- **trig1**/**trig2** can be used to arm the analyzer)

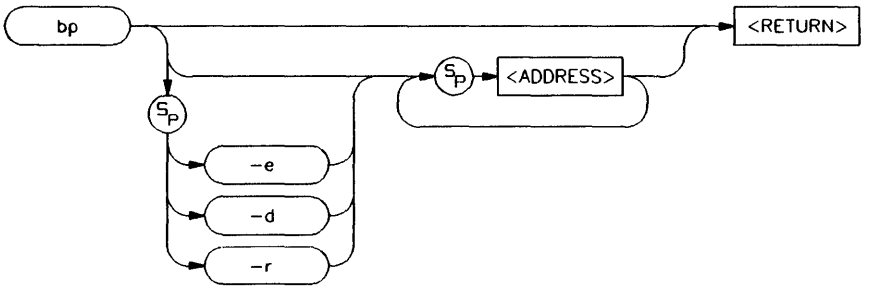
tgout (specifies which of the **trig1**/**trig2** signals are to be driven when the analyzer trigger is found)

bp

Summary

Insert or modify software breakpoints

Syntax



Function

The **bp** command is used to insert, delete, display, or modify the status of software breakpoints.

Note

Not all emulators support software breakpoints. Refer to the *Emulator User's Guide* supplied with your particular emulator to determine whether or not software breakpoints are supported.

There are four different operations to maintain the software breakpoint table.

Inserting Breakpoints

Specifying only an address inserts the breakpoint instruction in memory and makes a breakpoint table entry corresponding to that address. If a software break instruction already exists at the address specified, an error message is generated and the current **bp** command is aborted.

Enabling Breakpoints

Enabling a breakpoint at a specified address causes the system to search the breakpoint table for that address; if it exists in the table, the breakpoint instruction is written to memory at the corresponding address.

Disabling Breakpoints

Disabling the breakpoint for a specified address again causes a search for a breakpoint table entry; if found, the original contents of the address (before the breakpoint was defined) are written to the corresponding memory location. The contents of the breakpoint table are unchanged, except to indicate that the particular breakpoint is now inactivated.

Note

When the breakpoint table is displayed with the **bp** command, the enable/disable status of each breakpoint is tested by reading the memory locations in question. If a software break instruction is found, the breakpoint is displayed as **enabled**; if not, the breakpoint is displayed as **disabled**.

Note

Software breakpoints should not be set, enabled, disabled, or removed while the emulator is running user code. Also, you should not disable the "breakpoints" break condition (**bc -d bp**) while the emulator is running user code.

If any of these commands are entered while the emulator is running user code, and the emulator is executing code in the area where the breakpoint is being modified, program execution may be unreliable.

The problem occurs when the software breakpoint instruction (or the original opcode) is partially written in the emulation memory location while the emulation processor is fetching from that location; an illegal opcode may result. This problem does not occur when breakpoints are in target RAM because the emulation processor breaks into the monitor to enable or disable software breakpoints.

Removing Breakpoints

Removing a breakpoint causes a search for a corresponding breakpoint table entry; if found, the original memory contents are written to the specified address, and the entry is removed from the breakpoint table.

When a software breakpoint instruction inserted by **bp** is executed by your program, it is removed from memory and marked **disabled** in the breakpoint table.

A status message is issued to **stdout** (see the **po** command) indicating that a software breakpoint was found.

If the emulator executes a software break instruction that was placed by you (either through your compiler or via memory modification) and not by the **bp** command, an "undefined breakpoint" error message is generated.

If the emulator is executing in the user program when you define or modify breakpoints, it may break into the monitor for each breakpoint defined or modified. Whether or not it will do this

depends on the location of the breakpoint in memory (breaks to the monitor are required if the location is in target RAM), and whether your particular emulator must break to the monitor for accesses to that memory type (breaks into the monitor are not necessary if the location is in dual-port emulation memory). If a break to the monitor is required, the emulator will return to user program execution after breakpoint definition or modification.

In general, you should only define software breakpoints at memory locations which contain user program instructions. If you set breakpoints at other locations, it is unlikely that they will ever be executed. The only exception to this might be in a case where you suspect that your program is jumping into a data block and attempting to execute code; setting a software breakpoint in this area will allow you to verify the problem (and stop a runaway program).

Remember that any operation which modifies memory or the memory map will alter the existing breakpoints. For example, if you load a new program in the same address range where breakpoints reside, the breakpoints will be destroyed. Changing the memory map will prevent the emulator from placing new breakpoints or enabling existing breakpoints.

You cannot define breakpoints until you have enabled them with the **bc -e bp** command. If you disable the software breakpoint feature with the **bc -d bp** command, the breakpoints currently defined will remain in the breakpoint table, but will be disabled and will remain in that state until the breakpoint feature is reenabled and the specified breakpoints are reenabled (**bc -e bp** and **bp -e <ADDRESS>**).

You may define a maximum of 32 software breakpoints.

Parameters

<ADDRESS> The **<ADDRESS>** parameter allows you to specify the address location where the software breakpoint is to be inserted. If you specify options **-d**, **-a**, or **-h**, then the address specifies the location of the breakpoint to be deleted, activated, or inactivated. For these

options, you may specify the character * as the address specifier, indicating that the operation is to be performed on all of the addresses present in the software breakpoint table.

Note

The default for the **<ADDRESS>** parameter is a hexadecimal expression, however, other numeric bases may be specified. Refer to the **<ADDRESS>** syntax pages and the *Emulator User's Guide* for your particular emulator for information on specifying address information.

Note

The memory access mode for writing breakpoints is set by the **mo** (mode) command; if the mode is set to byte access and an odd address location is specified, an invalid instruction may be inserted for processors that expect alignment of opcodes on even byte boundaries.

-r — Deletes the software breakpoint(s) at the addresses specified. If the address specified does not contain a breakpoint instruction, an error will be returned. When the breakpoint is deleted, the original memory contents are restored, then the address is removed from the breakpoint table.

-e — Enables (activates) the breakpoint(s) at the address(es) specified. This installs the necessary breakpoint instruction in memory. If the breakpoint is already enabled, no action is taken.

-d — Disables (deactivates or "hits") the breakpoint(s) at the address(es) specified. The breakpoints remain in the breakpoint

definition table and can be reset by using the **bp -e <ADDRESS>** command. If the breakpoint is already disabled, no action is taken.

Defaults

If no parameters are specified, the current status of all breakpoints is displayed. Upon powerup or **init** initialization, the breakpoint table is cleared and the breakpoint feature is disabled.

Examples

The following examples use the 68000 sample program from Appendix A.

Assume that you need to verify that the processor is reaching the COMMAND_A, OUTPUT, and LOOP routines at addresses 202c, 2052 and 2064 respectively. You can insert software breakpoints at these addresses and run the program to each successive breakpoint. First, you must enable the software breakpoint feature. Type:

```
M> bc -e bp
```

Now define the breakpoints at the start of each routine by typing:

```
M> bp 202c 2052 2064
```

You can view the current breakpoint settings by typing:

```
M> bp
```

You will see:

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 000202c #enabled
bp 0002052 #enabled
bp 0002064 #enabled
```

Note the headline that says "BREAKPOINT FEATURE IS ENABLED". If you disable the software breakpoint feature with **bc -d bp**, this will read "BREAKPOINT FEATURE IS DISABLED" and you will not be able to define any new breakpoints (those already defined are not removed by **bc -d bp**).

Now, run the program from the start and modify the command input area at address 3000 by typing:

```
M> r 2000
U> m 3000=41
```

Since the command input was "A", the program will reach the software breakpoint entered at 202c hex. You will see:

```
!STATUS  615! Software breakpoint: 000202c@sp
```

You can run the processor to the next breakpoint by typing:

```
M> r
```

This is possible because the breakpoint is removed from location 202c after it is "hit"; the original instruction is returned to 202c.

You will see:

```
!STATUS  615! Software breakpoint: 0002052@sp
```

Now run the processor to the last breakpoint by typing:

```
M> r
```

You will see:

```
!STATUS  615! Software breakpoint: 0002064@sp
```

If desired, you can run the processor from this breakpoint by typing:

```
M> r
```

To break back to the monitor, type:

```
U> b
```

Now look at the status of the breakpoint table. Type:

```
M> bp
```

You will see:

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 000202c #disabled
bp 0002052 #disabled
bp 0002064 #disabled
```

Since all of the breakpoints were "hit" (executed by the processor), they are now disabled. You can reenable the existing breakpoints by typing:

```
M> bp -e 202c 2052 2064
```

You could also type **bp -e *** to reenable all breakpoints in the table.

View the reenabled breakpoints by typing:

```
M> bp
```

You will see:

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 000202c #enabled
bp 0002052 #enabled
bp 0002064 #enabled
```

If you want to disable a particular breakpoint, for example, the one at location 202c hex, type:

```
M> bp -d 202c
```

You can see the changes in the table by typing:

```
M> bp
```

You will see:

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 000202c #disabled
bp 0002052 #enabled
bp 0002064 #enabled
```

Note that the breakpoint at location 202c is now listed as disabled. Run the processor and enter a "command" into the sample program by typing:

```
M> r 2000
U> m 3000=41
```

```
!STATUS  615! Software breakpoint: 0002052@sp
```

Notice that the first breakpoint encountered is the first one enabled in the breakpoint table, that is, the breakpoint at 2052 hex. The breakpoint at 202c hex was not found; since it is disabled, no software break instruction was inserted in the code.

Run to the next breakpoint by typing:

```
M> r
```

```
!STATUS  615! Software breakpoint: 0002064@sp
```

To see the current status of the breakpoint table, type:

```
M> bp
```

You will see:

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 000202c #disabled
bp 0002052 #disabled
bp 0002064 #disabled
```

Again, all breakpoints are disabled; the one at 202c through your explicit **bp -d 202c** command, the others from being "hit" during the program run.

To completely remove the breakpoint at 202c from the breakpoint table, type:

```
M> bp -r 202c
```

Now type:

```
M> bp
```

You will see:

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 0002052 #disabled
bp 0002064 #disabled
```

As you can see, the breakpoint at address 202c has been removed from the table.

You can reenable all breakpoints in the table by typing:

```
M> bp -e *
M> bp
```

You will see:

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 0002052 #enabled
bp 0002064 #enabled
```

Conversely, you can disable all breakpoints in the table by typing:

```
M> bp -d *
M> bp
```

You will see:

```
### BREAKPOINT FEATURE IS ENABLED ###
bp 0002052 #disabled
bp 0002064 #disabled
```

Finally, if you want to remove all breakpoints in the table, type:

```
M> bp -r *
M> bp
```

You will see:

```
  ### BREAKPOINT FEATURE IS ENABLED ###
M>
```

Note that the breakpoint feature is still enabled, though there are no breakpoints defined in the table. To disable the breakpoint feature, use the command **bc -d bp**.

Related Commands

bc (enable/disable breakpoint conditions (including **bp**))

cf (set instruction type used for software breakpoint (only available on some processors))

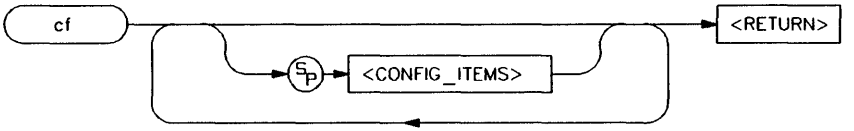
mo (defines memory access and display modes; the **bp** command uses the currently defined modes when writing software breakpoints into memory)

cf

Summary

Set emulator specific configuration items

Syntax



Function

The **cf** command allows you to modify various emulator specific configuration parameters. For example, **cf** will allow you to specify whether the clock source is in the user system or in the emulator. Each emulator has its own unique set of configuration items. For complete details, refer to the <CONFIG_ITEMS> syntax pages in the *Emulator User's Guide* for your particular emulator.

Parameters

Refer to the *Emulator User's Guide* for your emulator.

Defaults

If no parameters are specified, the current configuration settings are displayed. Refer to the *Emulator User's Guide* regarding the default settings for your emulator.

Examples

Refer to the *Emulator User's Guide* for your emulator.

Related Commands

help (you can get an on line display of the configuration items for a particular emulator by typing **help cf**. To obtain more information regarding a particular configuration item, type **help cf <config_item>**).

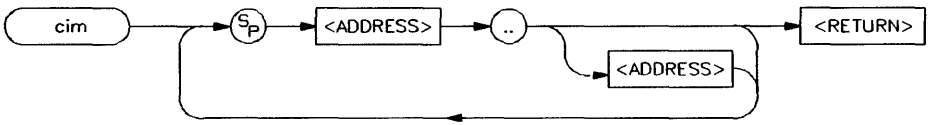
Notes

cim

Summary

Copy target system memory to emulation memory

Syntax



Function

The **cim** command allows you to copy an image of target memory into emulation memory. You might wish to do this for the following reasons:

You want to set software breakpoints to track down a problem. If your target memory is ROM, you cannot set breakpoints without programming new ROMs; however, you can easily copy the code to emulation memory and set or modify the breakpoints.

You want to test a code patch, and your target memory is in ROM as above.

You would like to perform execution coverage measurements with the **cov** command to find out how much of your software in the target is being accessed. (You need to move the code into emulation memory since the **cov** command only works with memory mapped as **eram** (emulation RAM) or **erom** (emulation ROM).)

Note that you may need to modify your memory map for this command to work correctly. Essentially, you need to map the addresses of the target system range you wish to copy to emulation memory space. Refer to the examples below.

Parameters

<ADDRESS>	Specifies the lower, and possibly upper, addresses of the range you want to copy. Although the default for <ADDRESS> is an expression in which the default base is hexadecimal, certain emulators allow specification of additional address items such as function codes. (You could also specify an equate defined with the **equ** command.) Refer to the *Emulator User's Guide* for your particular emulator for further information on address specification.
..	The two periods ".." are used as a separator between the lower and upper address boundaries of the range to be copied. Note that no additional spaces are inserted; if they are, a error message is generated. You can use "<ADDRESS>.." to specify the range from that address through the next 127 bytes.

Defaults

None; at least one address range must be specified; both the lower boundary and upper boundary can be the same; but in any case, the upper boundary must be greater than or equal to the lower boundary.

Examples

If you have target system ROM from 2000 hex to 2fff hex, and you would like to insert a breakpoint at 2010 hex, type the following:

```
R> map
```

You might see something similar to the following (dependent on your system memory map):

```
# remaining number of terms    : 5
# remaining emulation memory   : 1f800h bytes
map  001000..001fff     tram    # term 1
map  002000..002fff     trom    # term 2
map  other tram
```

Now you can delete the target ROM mapper term and substitute an emulation memory ROM term.

Type:

```
R> map -d 2
R> map 2000..2fff erom
```

To view the modified map, type:

R> **map**

You will see:

```
# remaining number of terms    : 5
# remaining emulation memory   : 1e800h bytes
map  001000..001fff     tram    # term 1
map  002000..002fff     erom    # term 2
map  other tram
```

Note that the emulator will now direct references to addresses 2000 through 2fff to emulation memory and ignores the presence of target system memory within that range. However, the **cim** command explicitly ignores the map; all **cim** address references refer to target system memory ranges. Now copy the code resident in the target ROM to emulation ROM by typing:

```
M> cim 2000..2fff
```

Now you can set the breakpoint by typing:

```
M> bc -e bp
M> bp 2010
```

Note

The memory addresses copied from the target system must have equivalent addresses already mapped to emulation memory before the **cim** command is executed; otherwise, an error message will be generated.

Related Commands

map (allows you to define the location and type of various memory address ranges)

cov (allows you to measure the percentage of memory locations accessed by your program within a particular range)

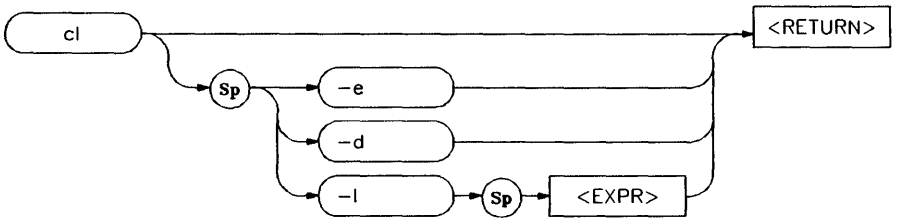
Notes

cl

Summary

Control command line editing

Syntax



Function

You can enable command line editing to include the ability to manipulate command text lines.

Command line editing has two typing modes. The normal command entry is input mode. The input mode functions like normal (canonical) command entry. The control mode allows command modification.

Refer to the appendix on "Command Entry" for more information about command line editing.

Parameters

-d	This option disables command line editing.
-e	This option enables command line editing.
-l	This option allows you to set the column length for the command line. This value can be from 40 to 132 columns.

Defaults

Command line editing is disabled.

Examples

To set the number of columns in the command line to 80, enter:

```
cl -l 80
```

With command line editing enabled, to add text to the previously executed command, enter:

```
<ESC> k
A
<additional text>
```

To display on-line help information for the **cl** command, enter:

```
help cl
```

The result on screen resembles:

```
cl - set or display command line editing mode

  cl               - display command line editing mode
  cl -e            - enable command line editing mode
  cl -d            - disable command line editing
  cl -l <columns> - number of columns for command line

--- VALID <columns> SELECTIONS ---
  range 40 to 132

--- Editing Mode Commands ---
  <ESC>- enter command editing mode
  i  - insert before current character   a - insert after current character
  A  - append to end of line             x - delete current character
  dd - delete command line               D - delete to end of line
  $  - move cursor to end of line        0 - move cursor to start of line
  ^  - move cursor to start of line      h - move left one character
  l  - move right one character          k - fetch previous command
  j  - fetch next command                r - replace current character

  /<string>  - find previous command in history matching <string>
  n  - fetch previous command matching <string>
  N  - fetch next command matching <string>
```

Related Commands

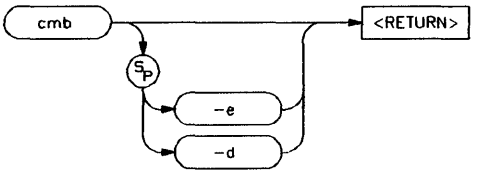
none

cmb

Summary

Enable/disable CMB interaction

Syntax



Function

The **cmb** command allows you to enable or disable interaction on the CMB (Coordinated Measurement Bus). The CMB allows you to make complex measurements involving cross-triggering of multiple HP 64700 analyzers and other HP 64000 system instruments, and synchronous emulator runs and breaks.

The **cmb** command only affects the ability for multiple emulators to run or break in a synchronized fashion; the analyzer trigger capability is unaffected by the **cmb** command.

Interaction Enabled

When interaction is enabled via the **cmb -e** command, the emulator will run code beginning at the address specified via the **rx** command when the CMB /EXECUTE (/ means active low) pulse is received.

The CMB READY line is driven false while the emulator is running in the monitor. The line goes to the true state whenever execution switches to the user program.

Note

Notice that if the **rx** command is given, CMB interaction is enabled just as if a **cmb -e** command was issued. Refer to the syntax pages for the **rx** command for further information.

Interaction Disabled

When interaction is disabled via the **cmb -d** command, the emulator ignores the actions of the /EXECUTE and READY lines. In addition, the emulator does not drive the READY line.

Parameters

-e	The **-e** option enables interaction between the emulator and the Coordinated Measurement Bus.
-d	The **-d** option disables interaction between the emulator and the Coordinated Measurement Bus.

Defaults

If no options are supplied, the current state of CMB enable/disable is displayed.

Examples

To view the current state of CMB interaction, type:

```
M> cmb
```

You will see:

```
cmb -d  #cmb currently disabled
```

To enable CMB interaction, type:

```
M> cmb -e
```

To disable interaction, type:

```
M> cmb -d
```

Related Commands

rx (allows you to specify the starting address for user program execution when the CMB /EXECUTE line is asserted)

tx (controls whether or not the emulation analyzer is started when the /EXECUTE line is asserted)

x (pulses the /EXECUTE line, initiating a synchronous execution among emulators connected to the CMB and enabled)

Also, refer to the *Coordinated Measurement Bus User's Guide* for further information on CMB operation.

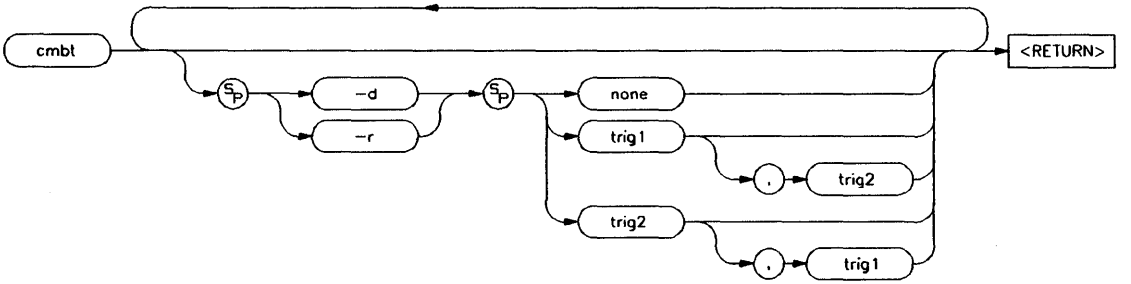
Notes

cmbt

Summary

Specify drivers and receivers of the CMB trigger signal

Syntax



Function

The **cmbt** command allows you to specify which of the internal **trig1**/**trig2** trigger signals will drive and/or receive the rear panel CMB (Coordinated Measurement Bus) trigger. You can specify the signals individually, as an ORed condition for drive, or as an ANDed condition for receive; or, you can specify that the signals are not to be driven and/or received.

You use this command to trigger other HP 64700 analyzers and possibly HP 64000 system instruments. For example, you may wish to start a trace on another HP 64700 analyzer when the analyzer in this emulator finds its trigger; or, you may wish to do the converse and trigger the analyzer in this emulator when another emulation analyzer finds its trigger.

Parameters

-d — The **-d** parameter causes the CMB to drive the trigger signals, trig1 and trig2, to the emulator's internal analyzer.

-r	The **-r** parameter causes the CMB to receive the trigger signals, trig1 and trig2, from the analyzer.
none	If you specify **none** with the **-d** option, then the CMB trigger signal will not drive either of the analyzer triggers. If you specify**none** with the **-r** option, the rear panel CMB will not receive trig1 or trig2 from the internal analyzer.
trig1	If **trig1** is specified, then the internal "trig1" signal will drive or receive the CMB trigger signal, depending on whether you specified the **-d** or **-r** option.
trig2	If you specify **trig2**, then the internal "trig2" signal will drive or receive the CMB trigger signal, depending on whether you specified the **-d** or **-r** option.

Note

You can also specify that both the **trig1** and **trig2** signals are to be driven and/or received. To do this, place a comma between the two signals on the command line.

Defaults

If no options are specified, the current setting of **cmbt** is displayed. Upon powerup, **cmbt** is set to **cmbt -d none -r none.**

Examples

To view the current **cmbt** setting, type:

```
M> cmbt
```

You will see:

```
cmbt -d none -r none
```

If you want to trigger the analyzer in another HP 64700 emulator hooked to the CMB, you might do the following:

```
M> tcf -e
M> tg addr=2000
M> tgout trig1
M> cmbt -d none -r trig1
```

By specifying this command sequence, the other HP 64700 analyzer will receive a trigger signal from its CMB when the emulation processor in this HP 64700 reaches the trigger pattern of address=2000.

To set the other HP 64700 analyzer to break to monitor upon receiving the CMB trigger, use the following command sequence:

```
M> cmbt -r trig1
M> bc -e cmbt
```

You might want to have an external instrument arm the analyzer in one emulator which then arms a second analyzer attached through the CMB. The second emulator then breaks to monitor when it finds its trigger condition. Use the following command sequence in the first emulator:

```
M> bnct -d trig1 -r none
M> tarm =trig1
M> tsq -i 3
M> tif 1 arm
M> tif 2 addr=2000
M> tgout trig2
M> cmbt -d trig2 -r none
```

Now, on the second emulator, type:

```
M> cmbt -d trig1 -r none
M> tarm =trig1
M> tsq -i 3
M> tif 1 arm
M> tif 2 addr=2018
M> tgout trig2
M> bc -e trig2
```

Note

You should not set up an analyzer in an emulator to both drive and receive the same trigger signal. For example, if you issued the commands **tg arm**, **tarm =trig1**, **tgout trig1**, and **cmbt -d trig1 -r trig1**, then the analyzer **trig1** signal will become latched in a feedback loop and will remain latched until the loop is broken. To break the loop, you must first disable the signal's source, then momentarily disable either the drive or receive function. In this case, the commands **tgout none** and **cmbt -d none** will break the loop.

Related Commands

bc (break conditions; can be used to specify that the emulator will break into the emulation monitor upon receipt of one of the **trig1/trig2** signals)

bnct (BNC trigger; used to specify which internal signals will be driven or received by the rear panel BNC connector)

cmb (Used to enable or disable interaction on the CMB. This does not affect whether measurement instruments can exchange triggers over the CMB; it only controls run/break interaction between multiple emulators)

tarm (analyzer trace arm; used to specify arming (begin to search for trigger) conditions for the analyzer -- **trig1/trig2** can be used to arm the analyzer)

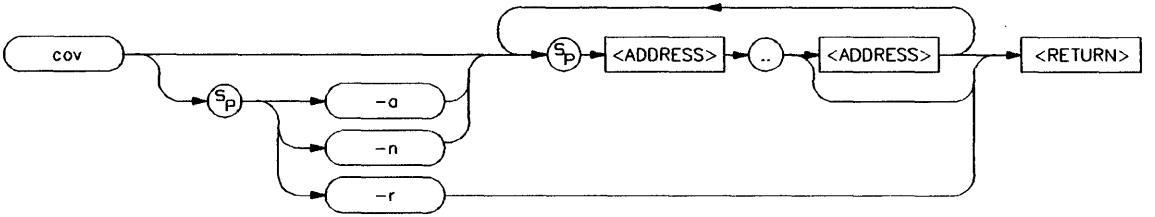
tgout (specifies which of the **trig1/trig2** signals are to be driven when the analyzer trigger is found)

cov

Summary

Measure percentage of memory locations accessed

Syntax



Function

The **cov** command allows you to measure the percentage of memory locations accessed within a certain range. Each memory location has a flag indicating whether or not it has been accessed; the flag is set automatically when the address of that location appears on the emulation processor bus.

The percentage accessed is calculated by dividing the number of different locations accessed (no location is counted more than once) by the number of different locations within the range and multiplying by 100.

Coverage measurements can only be performed on address ranges mapped to emulation memory. If you wish to perform coverage measurements on target system memory ranges, you can remap memory and copy the target system memory to emulation memory using the **cim** command.

Parameters

-r — The **-r** parameter resets the results from the previous coverage measurement and sets all of the coverage data bits to the NOT-ACCESSED state.

The **-r** option cannot be used in conjunction with other parameters.

-a — The **-a** option produces a list of locations within the specified memory range that were accessed.

-n — The **-n** option produces a list of locations within the specified memory range that were not accessed; the percentage measurement given is in terms of locations not accessed..

<ADDRESS> — Specifies the lower, and possibly upper, memory address boundaries for the coverage measurement. The default is a hexadecimal number; other bases may be specified. Certain emulators allow additional processor specific addressing information for <ADDRESS>; refer to the *Emulator User's Guide* for your particular emulator for further information.

Multiple address ranges for coverage testing can be specified; a space character must be included between each range specification.

.. — The separator between the lower and upper address boundaries is two periods." Notice that no additional spaces are inserted. You can use "<ADDRESS>.." to specify a range from the address through the next 127 bytes.

Note 

Overlapping ranges in the **cov** command will result in an incorrect coverage percentage.

Defaults

At least one address range must be specified. All of the coverage flags are reset at powerup or by **init**.

Examples

If you would like to measure the memory coverage of the 68000 sample program shown in appendix A, type the following after loading the program:

```
M> cov -r
```

Note

You should make a habit of resetting the coverage information before making measurements. This is because any activity which accesses memory, even loading memory with your programs using modify memory, will cause the coverage bits to be set for those memory locations, leading to inaccurate measurement results.

```
M> r 2000
U> m 3000=41
```

This memory modification command causes the program to run some of the output routines; it will access more memory locations. Now display the coverage:

```
U> cov 2000..2071
```

You will see:

```
percentage of memory accessed:  % 68.4
```

To reset the coverage data (which clears the coverage memory in preparation for another measurement), type:

```
U> cov -r
```

To display the coverage on the overhead routines of the program (INIT, CLEAR and READ_INPUT), type:

```
U> cov 2000..2071
```

You will see:

```
percentage of memory accessed:  %  8.7
```

You should generally reset the coverage information between measurements for the most accurate results. (The only exception might be if you just need to specify additional address ranges for a measurement.)

Type:

```
U> cov -r
U> m 3000=43
U> cov -n 1000..1038 2000..2071
```

You will see:

```
# coverage list  -  list of address ranges NOT accessed
0001000..0001029
0002000..000200b
000202c..0002047

percentage of memory NOT accessed:  % 47.9
```

Note that the percentage is expressed in terms of memory locations which were not accessed.

You can also display a list of the locations actually accessed within the range. Type:

```
U> cov -a 1000..1038 2000..2071
```

You will see:

```
# coverage list  -  list of address ranges accessed
000102a..0001038
000200c..000202b
0002048..0002071

percentage of memory accessed:  % 52.0
```

Related Commands

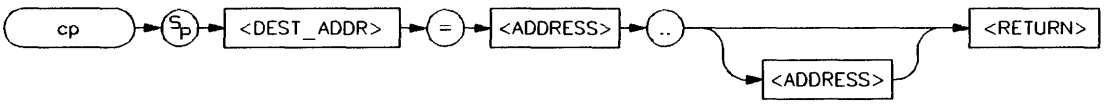
cim (allows you to copy target system memory images into emulation memory for coverage measurements)

cp

Summary

Copy memory blocks

Syntax



Function

The **cp** command allows you to copy a block of data from one region of memory to another. For example, you might want copy a data table in your program to a buffer space so you can try some of your algorithms for processing data in that buffer.

When **cp** is executed, the data from the specified range is copied to the destination address, with the lower boundary data going to the destination address, lower boundary + 1 to destination + 1, and so on until the upper boundary of the source range is copied. If the source or destination addresses reside within the target system, the emulator will break to the background monitor and will return to foreground after the copy is completed.

If memory mapped as guarded is encountered in the source or destination range during the copy, the command is aborted; however, all locations modified prior to accessing guarded memory are left in the modified state.

Parameters

<DEST_ADDR>

Specifies the lower boundary of the destination range. The processor specific conventions for <ADDRESS> can be used for complete address specification including function codes or segmentation. Refer to the *Emulator User's Guide* for your particular emulator for details.

<ADDRESS>	Specifies the lower, and possibly upper, memory address boundaries of the source range to be copied. The default is a hexadecimal number; other bases may be specified. Certain emulators allow additional processor specific addressing information for <ADDRESS>; refer to the *Emulator User's Guide* for your particular emulator for further information.
..	The separator between the lower and upper address boundaries is two periods (..). Notice that no additional spaces are inserted. You can use "<ADDRESS>.." to specify a range from the address through the next 127 bytes.

Defaults

Exactly one address range must be specified.

Examples

You can use the **cp** command to move the data area of the 68000 sample program (from Appendix A) from a base address of 1000 hex to a base address of 5000 hex. First, let's look at the original data area. Type:

```
M> m -db 1000..1038
```

You will see:

```
001000..00100f      00 00 30 00 00 00 40 00 54 48 49 53 20 49 53 20
001010..00101f      4d 45 53 53 41 47 45 20 41 54 48 49 53 20 49 53
001020..00102f      20 4d 45 53 53 41 47 45 20 42 49 4e 56 41 4c 49
001030..001038      44 20 43 4f 4d 4d 41 4e 44
```

Now you can copy the block to a base address of 5000 hex. Type:

```
M> cp 5000=1000..1038
```

To view the new block, type:

```
M> m 5000..5038
```

You will see:

```
005000..00500f      00 00 30 00 00 00 40 00 54 48 49 53 20 49 53 20
005010..00501f      4d 45 53 53 41 47 45 20 41 54 48 49 53 20 49 53
005020..00502f      20 4d 45 53 53 41 47 45 20 42 49 4e 56 41 4c 49
005030..005038      44 20 43 4f 4d 4d 41 4e 44
```

The new block is identical to the old.

Related Commands

cim (copies a memory image from the target system to emulation memory)

m (allows you to display or modify memory locations or ranges)

map (used to define the type and location of memory used by the emulator)

ser (used to search memory ranges for a specific set of data values)

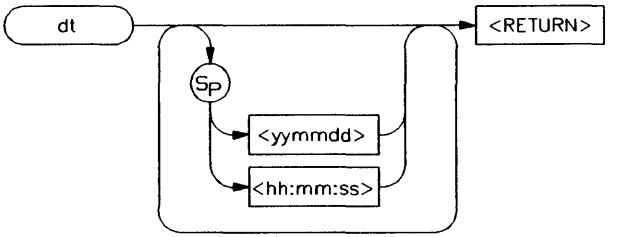
Notes

dt

Summary

Set or display system date/time

Syntax



Function

The **dt** command allows you to set or display the current date and time stored by the HP 64700 series emulators.

Note

The emulator system date & time clock is reset when power is cycled.

Parameters

<yymmdd>	This variable sets the date. **yy** are the last two digits of the current year; **mm** specify the current month, and **dd** specify the day of the month.

Note

If **yy** is greater than 50, the year is assumed to be in the 20th century (in other words, **19yy**). If **yy** is less than 50, the year is assumed to be in the 21st century (in other words, **20yy**).

<hh:mm:ss> This variable sets the time in 24 hour format. **hh** specify the hour, **mm** specify the minutes, and **ss** specify the seconds. Notice that the only difference between the date and time variables is the presence of colons; therefore, if you forget the colons while trying to reset the time, you will actually change the date setting.

Defaults

If no parameters are specified, the current date and time settings are displayed.

Examples

To display the current date and time settings at emulator powerup, type:

```
M> dt
```

You will see:

```
January 01, 1988  0:00:21
```

To set the date to August 18, 1987, type:

```
M> dt 870818
```

To set the date to August 18, 1987 and the time to 11:05:00, type:

```
M> dt 870818 11:05:00  (order of the two
arguments is not significant)
```

Related Commands

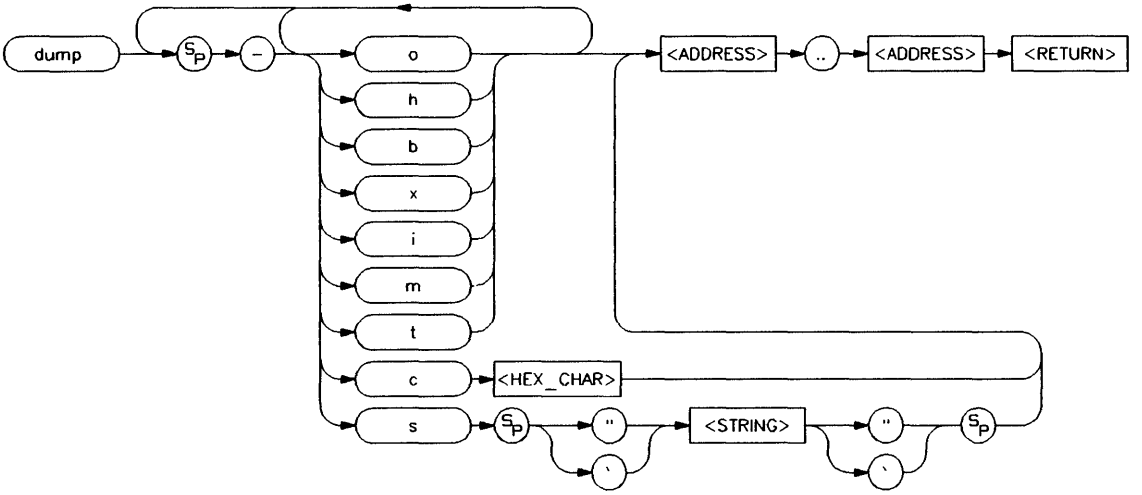
None

dump

Summary

Dump memory to a host file

Syntax



Function

The **dump** command allows you to dump the contents of emulation and/or target system memory to a host file. The contents can be dumped in HP, Tektronix hex, Intel hex, and Motorola S-record formats by specifying various options on the command line.

If you are uploading the file in HP file format using the HP 64000 **transfer** software, record checking is performed automatically by the **transfer** protocol.

Parameters

-o

The **-o** option indicates that the memory records dumped will be sent to the **other** port from the one that issued the command.

This is primarily used when operating in transparent mode where a terminal is connected to the command port and a host computer is connected to the other port.

-s

The **-s** option, along with a **<STRING>** parameter, sends the given <STRING> out the **other** port and dumps the records to the host out that port (as in the **-o** option). The <STRING> is intended to initiate the transfer receive process on the host computer. The HP 64700 Emulator never enters transparent mode.

<STRING>

<STRING> specifies the command string to be sent to the host computer to initiate the transfer receive process. The string must be enclosed in either single open quotes (‘) or in double quotes (").

Note

Many keyboards (and printers) represent the single open quote mark as an accent grave mark. In any case, the correct character is ASCII 60 hexadecimal. The correct double quote character is ASCII 22 hexadecimal.

-h

The **-h** option indicates that the memory contents will be dumped in HP absolute file format.

-b

Specifying the **-b** option indicates that the records will be sent in binary; this is only valid with **-h** (HP file format).

-x

If you specify **-x**, the records will be sent in hexadecimal; this is only valid with the **-h** option (HP file format).

-i	Specify the **-i** option if you need to have the file transferred in Intel hex record format. Note that the various options for HP file format transfer (such as **-x**, **-b**, and **-e**) are invalid with this format.
-m	Specify the **-m** option if you need to have the file transferred in Motorola S-record format.
-t	Specify the **-t** option if you need to have the file transferred in Tektronix hex format.
-c	Specifying -c along with an ASCII hexadecimal character indicates that the character specified should be sent to the host at the end of the file upload.
<HEX_CHAR>	<HEX_CHAR> is an ASCII character to be sent to the host at the end of the upload process. The character is used to close the host file which is receiving the uploaded data.
<ADDRESS>	Specifies the lower, then upper, address boundaries of the memory range to be dumped. The default is a hexadecimal number; other bases and expressions may be supplied. Refer to the <EXPR> syntax pages for details. In addition, many microprocessors allow special address information such as segmentation or function codes to be specified; refer to the <ADDRESS> syntax pages in the *Emulator User's Guide* for details.

Defaults

None; a file format and address range must be specified.

Examples

To dump the contents of memory range 2000 hex through 20ff hex to a file named "intelfile" on a host, do the following:

First, establish transparent mode communications with the host system (that is, one port of HP 64700 to terminal; other port to host; use the **xp** command to log on to host then return to command mode).

Now type:

```
M> dump -ioc04 2000..20ff<RETURN>
cat > intelfile<ESC>*
```

This will send the contents of memory from locations 2000 through 20ff hex to the host in Intel hex record format; the records will be dumped to the "other" port (not the command port); at the end of the upload process, the character 04 hex (<CTRL>-D) will be sent to the host to close the file. Note that a <RETURN> is not entered after the **cat intelfile** command; the **<ESC>*** (where * is the transparent mode escape character) issues the return to the host automatically.

To dump memory locations 1000..2fff hex in HP file format using the HP **transfer** software, type:

```
M> dump -hox 1000..2fff<RETURN>
transfer -fax DUMPFILE::absolute<ESC>*
```

Again, no return is entered after the host command sequence. Here, the records are dumped in HP file format to the "other" port. The file will be named **dumpfile.X** on the host by the **transfer** process.

You can also use the **-s** option of the **dump** command to enter the **transfer** string directly. Type:

```
M> dump -hxs 'transfer -fax
DUMPFILE::absolute' 1000..2fff
```

To simplify repetitive dumps, you can define a macro named **dmp** as follows:

```
M> mac dmp={dump -hxs 'transfer -fax
DUMPFILE::absolute' 1000..2fff}
```

Now, to dump records to the host, simply type:

```
M> dmp
```

Note

The HP 64000 format ".X" file created with a "dump -hox" command has records that contain 136 fewer bytes of data than the file format standard allows. Because of this, HP 64000 format ".X" files which are created with the **dump** command may take longer to be processed by consumers of the ".X" file (depending on how the consumer processes sequential records).

Related Commands

load (used to load emulation memory from a host computer file)

xp (may be used to establish communications with a host computer)

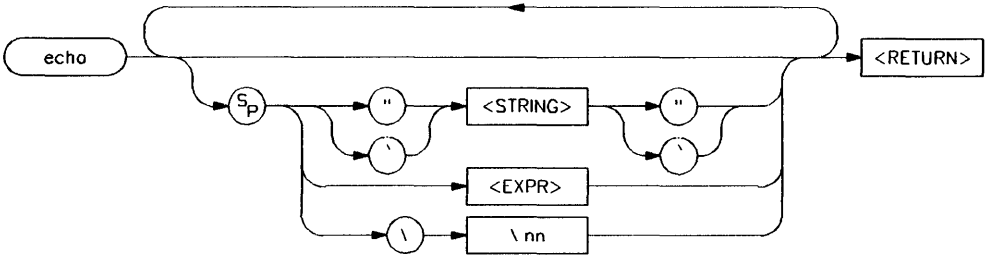
Notes

echo

Summary

Echo character strings or expressions

Syntax



Function

The **echo** command allows you to display ASCII strings or the results of evaluated expressions on the standard output device. You must enclose strings in single open quote marks (‘) (ASCII 60 hex) or double quotation marks (") (ASCII 22 hex). A string not enclosed in delimiters will be evaluated as an expression and the result will be echoed. In addition, you may supply a backslash with a two digit hex constant; the corresponding ASCII character(s) will be echoed.

Echoing strings or ASCII characters is particularly useful within macros, command files, and repeats where you wish to prompt the user to perform some action during a "wait for any keystroke" command (see syntax for **w**). The expression capability is useful as a quick calculator.

If you use **echo** in combination with the **po** command, the desired information may be directed to another port. Generally, you should do this only by combining options on the command line or with the **macro** or **rep** command; these will allow you to correct typing errors before a mistake occurs.

Note that all options may combined within the same echo command as long as they are separated by spaces.

Parameters

<STRING>	Any set of ASCII characters enclosed between single open quote marks (‘), or double quotes ("). Since the command buffer is limited to 256 characters, the maximum number of characters in a string is 248.

Note 

Many keyboards (and printers) actually represent the single open quote mark (ASCII 60 hexadecimal) as an accent grave mark. The correct character in any case is the one encoded as ASCII 60 hexadecimal. The correct double quotation mark is ASCII 22 hexadecimal.

Note 

A character which is used as a delimiter cannot be used within the string. For example, the string "**Type "C""** is incorrect and will return an error. The string ‘**Type "C"**‘ is correct.

<EXPR>	A valid expression (refer to the expression syntax pages for descriptions of valid expressions). The expression will be evaluated and the result will be echoed. Note that no delimiters are used to define the start and end of the expression.
<nn>	"nn" is the hex code for any valid ASCII character. More than one character can be echoed with a single command; each "nn" must be preceded by a backslash. A total of 62 ASCII characters can be represented within a single **echo** command.

This capability is particularly useful for sending non-displaying control characters to a terminal; refer to the examples below.

Defaults

Echo nothing.

Examples

To echo the string "Set S1 to OFF" to the standard output, type the following:

```
M> echo "Set S1 to OFF"
```

OR

```
M> echo 'Set S1 to OFF'
```

You will see:

```
Set S1 to OFF
```

Alternatively, you could use the ASCII character evaluation capability to do the same thing by typing the following:

```
M> echo \53 \65 \74 \20 \53 \31 \20 \74 \6f \20
\4f \46 \46
```

You will see:

```
SET S1 to OFF
```

However, a more useful application of the backslash option is to send a terminal control characters. Type:

```
M> echo \1b "H" \1b "J" \1b "&dBSet S1 to OFF"
```

The above command sends "<ESC>H<ESC>J<ESC>&dB Set S1 to OFF" to the terminal. On an HP 2392A this homes the cursor, clears the screen, sets the video mode to inverse video, and writes the message "Set S1 to OFF". Therefore, the user would see the message "Set S1 to OFF" in inverse video at the upper left hand corner of an otherwise blank screen.

You might combine this with a macro command as part of a procedure. For example, type:

```
M> mac PROMPT={echo "Set S1 to OFF";w}
M> PROMPT
```

You will see:

```
Set S1 to OFF
Waiting for any keystroke...
```

If you want to start a process on a host computer represented by the command **startproc**, then type:

```
M> po -o B;echo "startproc";po -o A
```

Note 

The example above assumes that you are using a terminal as the standard input and standard output on port A, the host computer is connected to port B, and communications have been established previously through a set of **xp** commands.

To calculate the value of the expression (1f + 1e), type:

```
M> echo 1f+1e
```

You will see:

```
03dh
```

See the syntax pages for **expr** for more details on construction of various expressions.

Note 

When using **echo** to calculate results of expressions, be aware that all operations are carried out on 32-bit two's complement signed integers. Results greater than 32 bits are truncated.

Related Commands

expr (details on what constitutes valid expressions)

mac (grouping a set of commands under a label for later execution)

rep (grouping a set of commands for immediate repetition)

w (wait command, allows user specified delays)

po (allows definition of ports)

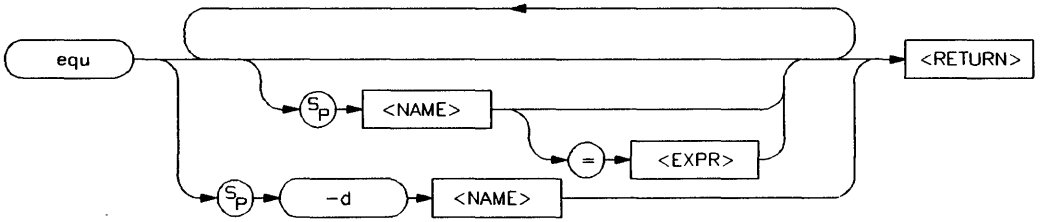
Notes

equ

Summary

Equate names to expressions

Syntax



Function

The **equ** command allows you to equate arithmetic values with names that you can easily remember; these names can then be used in other commands to reference the value. This is useful in defining trigger patterns for the analyzer and in other applications.

Multiple equates may be defined on the same command line, separated by a space.

Note

Each equate is translated to its actual value at the time of command entry. For example, if you specify an equate **count=21h**; and an expression **start=2000h**, then the command **tg addr=start count** will be entered into the system as **tg addr=start 33**. At this point, redefining the value of **addr** or **count** would not change the address expression or the occurrence counter for the trigger.

Parameters

<NAME> — You use <NAME> to assign a character string to the expression. <NAME> must be an alphanumeric designator no greater than 31 characters in length, beginning with an alpha character or underscore and including only alphanumeric characters or underscores thereafter. If <NAME> is specified without an expression, then the existing definition for that name is displayed. If <NAME> is specified as *, and the **-d** option is not given, then the definitions for all equates is displayed. However, if **-d** is supplied, then the equate table is cleared.

Note 

Certain HP 64700-Series Emulators may predefine equates, such as those which equate names to certain processor status bit patterns. You should be careful not to delete these equates, as they are useful in specifying analyzer trace qualifiers.

<EXPR> — An arithmetic expression to be assigned to <NAME>. The default is a hexadecimal number. Refer to the <EXPR> syntax pages in this manual for further details.

Note 

The combination of a single **equ** command with all names and expressions cannot exceed 255 characters. The number of equates and symbols that may be defined is limited only by available system memory; thus, it is dependent on the number of macros defined and on any emulator control code loaded by a high level software interface for the emulator (such as the HP 64700 PC Interface).

-d	The **-d** option allows you to delete an existing equate. If you specify **-d** and <NAME>, then the named equate is deleted. If <NAME> is given as *, then all equates are deleted.

Defaults

If no parameters are specified, then the current table of all equates is displayed. If <NAME> is specified, then only the equate for that particular name is displayed.

Examples

If you are working with the 68000 sample program in Appendix A, you can predefine some equates to make it easier to set up analyzer and run specifications.

For example, to equate the string "start" to the address value 2000 hex, type:

```
M> equ start=2000
```

You can also make labels for arrays by using the power of the expression specifications. For example, to define equates for the message labels in the 68000 emulator tutorial program (included in the appendix), type:

```
M> equ msgtbl=1008h msgsize=17T
M> equ msga=msgtbl+0*msgsize
msgb=msgtbl+1*msgsize invmsg=msgtbl+2*msgsize
```

To see the newly defined list of equates, type:

```
M> equ
```

You will see:

```
### Equates ###
equ cyc6800=0xxxxx0xxy
equ dma=0xx011xxxy
equ grd=0xxxxxxxy
equ intack=0xx111xxxy
equ invmsg=102ah
equ msga=1008h
equ msgb=1019h
equ msgsize=11h
equ msgtbl=1008h
equ read=0xxxxxx1xy
equ start=2000h
equ supdata=0xx101xxxy
equ supprog=0xx110xxxy
equ userdata=0xx001xxxy
equ userprog=0xx010xxxy
equ write=0xxxxxx0xy
equ wrrom=0x0xxxx0xy
```

(Note that the HP 64700 Emulator for the 68000 predefines some equates for use in tracing various status conditions.)

These can now be used to create analyzer trigger expressions. For example, you may want to have the analyzer trigger on the program start plus an access to any one of the messages. To do this, type:

```
M> tinit
```

(Initializes the analyzer.)

```
M> tcf -c
```

(Sets trace configuration to complex configuration.)

```
M> tsq -t 3
```

(Sets trigger term as third sequencer term.)

```
M> tpat p1 addr=start
```

(Defines p1 to be the equate "start.")

```
M> tpat p2 addr=msga
```

(Defines p2 to be the equate "msga.")

```
M> tpat p3 addr=msgb
```

(Defines p3 to be the equate "msgb.")

```
M> tpat p4 addr=invmsg
```

(Defines p4 to be the equate "invmsg.")

```
M> tif 1 p1 2
```

(Jump from term 1 to term 2 upon finding "start.")

```
M> tif 2 p2|p3|p4 3
```

(Jump to trigger term if "msga","msgb","invmsg" accessed.)

```
M> trng addr=1008..1038
```

(Defines a range specifier of the message area.)

```
M> tsto r
```

(Analyzer stores only accesses to the message area.)

```
M> tp s
```

(Positions trigger at beginning of trace list.)

```
M> t
```

(Begins trace.)

```
M> r start
```

(Starts program run at equate "start.")

The next three commands successively enter "command A", "command B" and an "unrecognized command" into the program's input area.

```
U> m 3000=41
U> m 3000=42
U> m 3000=43
U> tl -1..52
```

You will see:

Line	addr,H	68000 Mnemonic	count,R	seq
-1	002000	2479 supr prog	---	+
0	001008	54 supr data rd byte	14.15 S	+
1	004000	54 supr data wr byte	0.400 uS	+
2	002068	0001 supr prog	0.400 uS	+
3	00206A	66F8 supr prog	0.400 uS	+
4	00206C	4EF9 supr prog	0.400 uS	+
5	002064	12D8 supr prog	0.600 uS	+
6	001009	48 supr data rd byte	0.800 uS	.
7	00100A	49 supr data rd byte	3.000 uS	.
8	00100B	53 supr data rd byte	3.000 uS	.
9	00100C	20 supr data rd byte	3.000 uS	.
10	00100D	49 supr data rd byte	3.000 uS	.
11	00100E	53 supr data rd byte	3.000 uS	.
12	00100F	20 supr data rd byte	3.000 uS	.
13	001010	4D supr data rd byte	3.000 uS	.
14	001011	45 supr data rd byte	3.000 uS	.
15	001012	53 supr data rd byte	3.000 uS	.
16	001013	53 supr data rd byte	3.000 uS	.
17	001014	41 supr data rd byte	3.000 uS	.
18	001015	47 supr data rd byte	3.000 uS	.
19	001016	45 supr data rd byte	3.000 uS	.
20	001017	20 supr data rd byte	3.000 uS	.
21	001018	41 supr data rd byte	3.000 uS	.
22	001019	54 supr data rd byte	2.701 S	.
23	00101A	48 supr data rd byte	3.000 uS	.
24	00101B	49 supr data rd byte	3.000 uS	.
25	00101C	53 supr data rd byte	3.000 uS	.
26	00101D	20 supr data rd byte	3.000 uS	.
27	00101E	49 supr data rd byte	3.000 uS	.
28	00101F	53 supr data rd byte	3.000 uS	.
29	001020	20 supr data rd byte	3.000 uS	.
30	001021	4D supr data rd byte	3.000 uS	.
31	001022	45 supr data rd byte	3.000 uS	.
32	001023	53 supr data rd byte	3.000 uS	.
33	001024	53 supr data rd byte	3.000 uS	.
34	001025	41 supr data rd byte	3.000 uS	.
35	001026	47 supr data rd byte	3.000 uS	.
36	001027	45 supr data rd byte	3.000 uS	.
37	001028	20 supr data rd byte	3.000 uS	.
38	001029	42 supr data rd byte	3.000 uS	.
39	00102A	49 supr data rd byte	3.375 S	.
40	00102B	4E supr data rd byte	3.000 uS	.
41	00102C	56 supr data rd byte	3.000 uS	.
42	00102D	41 supr data rd byte	3.000 uS	.
43	00102E	4C supr data rd byte	3.000 uS	.
44	00102F	49 supr data rd byte	3.000 uS	.
45	001030	44 supr data rd byte	3.000 uS	.
46	001031	20 supr data rd byte	3.000 uS	.
47	001032	43 supr data rd byte	3.000 uS	.
48	001033	4F supr data rd byte	3.000 uS	.
49	001034	4D supr data rd byte	3.000 uS	.
50	001035	4D supr data rd byte	3.000 uS	.
51	001036	41 supr data rd byte	3.000 uS	.
52	001037	4E supr data rd byte	3.000 uS	.

As you can see from the listing above, the program accessed all three of the program message areas.

You can remove equates from the table either individually or all at once. Type:

```
M> equ -d start
M> equ
```

You will see:

```
### Equates ###
equ cyc6800=0xxxxx0xxy
equ dma=0xx011xxxy
equ grd=0xxxxxxxy
equ intack=0xx111xxxy
equ invmsg=102ah
equ msga=1008h
equ msgb=1019h
equ msgsize=11h
equ msgtbl=1008h
equ read=0xxxxxx1xy
equ supdata=0xx101xxxy
equ supprog=0xx110xxxy
equ userdata=0xx001xxxy
equ userprog=0xx010xxxy
equ write=0xxxxxx0xy
equ wrrom=0x0xxxx0xy
```

Notice that the equate for the name **start** has been removed. Now type:

```
M> equ -d *
M> equ
```

You will see:

```
### Equates ###
```

Now all of the equates have been deleted.

You can also use equates for more subtle applications. For example, suppose you want to take 5 traces of the sample 68000 program, with the trigger at address 2010. You would like to have each trace numbered.

Enter the following commands:

```
M> tg addr=2010
M> equ c=0
M> mac numtrclist={t;w -m;equ c=c+1;echo
"trace # " c;tl}
M> r 2000
M> rep 5 tlist
```

You will see five trace lists, each sequentially numbered, displayed on screen. You could use this feature in combination with a host logging program or redirection of your terminal display to printer to continuously monitor operation of a system. (To further aid your troubleshooting, you could also display the date and time of each trace sample using the **dt** command.)

Related Commands

tg, tpat, tif, telif, and others. (**equ** provides an easy way to name expressions to use in setting up trigger or branch conditions)

r, m, bp (equates may be used to specify run addresses, memory addresses, or breakpoint addresses)

es

Summary

Display emulator status

Syntax

Function

The **es** command displays the current status of emulation activity. The following types of information may be displayed:

- processor status -- running/in monitor/reset
- slow bus cycle
- slow clock
- emulation halted due to halt input from target system or output from processor
- emulation in "wait" state due to input signal (ready, sync, DTACK) from target system
- emulation in monitor due to bus grant to the target system

The exact messages and information displayed varies slightly depending on the emulator in use.

The emulator will not break to the monitor to obtain information. Therefore, any information that can only be obtained while in the monitor will not be displayed if the emulator is not in the monitor.

Parameters

None.

Defaults

Does not apply.

Examples

These examples were constructed using the 68000 emulator, running the sample program from Appendix A.

```
                              M> es
M68000--Running in monitor

                              M> r 2000
                              U> es
M68000--Running user program

                              U> rst
                              R> es
M68000--Emulation reset

                              R> cf clk=ext
                              C> es
M68000--Slow clock
```

Related Commands

ta (allows you to display activity on emulation and external analyzer lines)

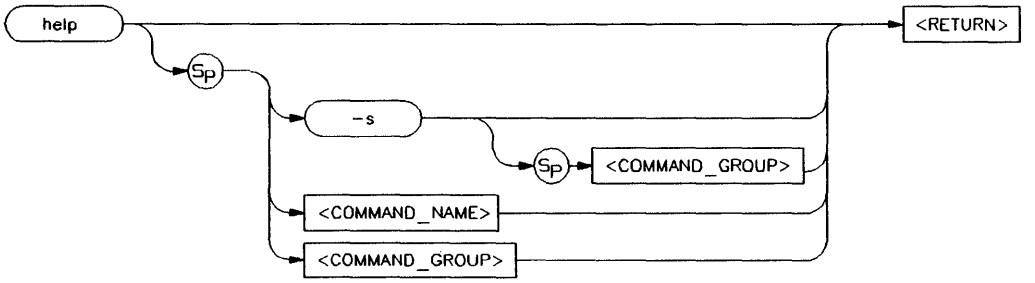
ts (allows you to display the current status of the emulation analyzer)

help,?

Summary

Display help information for commands

Syntax



Function

The **help** (?) command lets you display syntax, description and examples for any HP 64700 emulator Terminal Interface command. You may display a brief description for anything from a single command to command groups or the entire command set. Detailed information is available for single commands.

You may enter a question mark ? instead of typing help; it performs the same function.

Parameters

-s	This option switches in the abbreviated help mode; only the expanded name of each command is displayed next to the command.
<COMMAND_NAME>	If the name of an individual command is specified, only the detailed help information is displayed for that command.

<COMMAND_ GROUP>	Specifying the name of a command group lists the commands available within that group.

Note

If you specify "*" for <COMMAND_NAME> or <COMMAND_GROUP>, information for all commands will be displayed.

Defaults

The **help** command without any parameters provides a list of command groups.

Examples

To display general help information listing the command groups and information regarding the use of the **help** command, type:

```
M> help
```

You will see:

```
help  - display help information

  help <group>          - print help for desired group
  help -s <group>       - print short help for desired group
  help <command>        - print help for desired command
  help                  - print this help screen

--- VALID <group> NAMES ---
  gram      - system grammar
  proc      - processor specific grammar

  sys       - system commands
  emul      - emulation commands
  trc       - analyzer trace commands
  *         - all command groups
```

To display the short version of the help listing, which lists only the command groups and the commands available in the group, type:

```
M> ? -s
```

(Note that we typed the question mark symbol instead of help. You could use either with the same results.)

You will see:

```
sys    : ?, bnct, cmbt, dt, echo, equ, help, init, mac, po, pv, rep,
         stty, ver, w, x, xp
emul   : b, bc, bp, cf, cim, cmb, cov, cp, dump, es, io, load, m,
         map, mo, r, reg, rst, rx, s, ser
trc    : t, ta, tarm, tcf, tck, tcq, telif, tf, tg, tgout, th, tif,
         tinit, tl, tlb, tp, tpat, tpq, trng, ts, tsck, tsq, tsto, tx
```

To display the same listing of commands for only one of the command groups, type:

```
M> help -s emul
```

You will see:

```
emul   : b, bc, bp, cf, cim, cmb, cov, cp, dump, es, io, load, m,
         map, mo, r, reg, rst, rx, s, ser
```

You can display more information about each of the available memory commands by leaving out the -s flag. Type:

```
M> help emul
```

You will see:

```
emul - emulation commands
------------------------------------------------------------------------
b......break to monitor  cp.....copy memory       mo.....modes
bc.....break condition   dump...dump memory       r......run user code
bp.....breakpoints       es.....emulation status  reg....registers
cf.....configuration     io.....input/output      rst....reset
cim....copy target image load...load emul memory  rx.....run at CMB execute
cmb....CMB interaction   m......memory            s......step
cov....coverage          map....memory mapper     ser....search memory
```

Finally, to display specific information for the **m** command, type:

```
M> help m
```

You will see:

```
m - display or modify processor memory space

  m <addr>                - display memory at address
  m -d<dtype> <addr>      - display memory at address with display option
  m <addr>..<addr>        - display memory in specified address range
  m -dm <addr>..<addr>    - display memory mnemonics in specified range
  m <addr>..              - display 128 byte block starting at address A
  m <addr>=<value>        - modify memory at address to <value>
  m -d<dtype> <addr>=<value>        - modify memory with display option
  m <addr>=<value>,<value>          - modify memory to data sequence
  m <addr>..<addr>=<value>,<value> - fill range with repeating sequence

--- VALID <dtype> MODE OPTIONS ---
  w - display size is 2 byte(s)
  b - display size is 1 byte(s)
  l - display size is 4 byte(s)
  m - display processor mnemonics
```

Related Commands

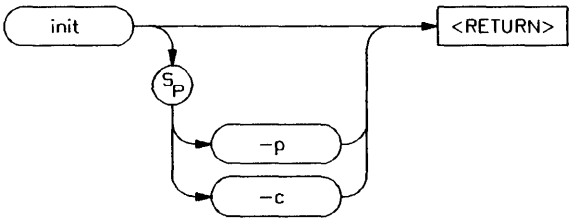
None.

init

Summary

Initialize the emulator

Syntax



Function

The **init** command allows you to re-initialize the emulator. Powerup, complete, and limited initializations are available through various options. In most cases you should only use this command if the emulator is not responsive to other commands. If you wish to change other configuration parameters without initializing the emulator, there are commands available for that purpose. Refer to the list under "Related Commands" at the end of this chapter.

Parameters

-p	The **-p** option causes a powerup initialization sequence. This initializes the operating system, data communications, emulation and analyzer boards, and runs extensive performance verification.
-c	The **-c** option causes a complete initialization sequence. Everything is initialized as defined by the powerup sequence with the exception of the performance verification.

Note 

The **init -c** and **init -p** commands cause a loss of system memory. If these commands are used in macros, commands that follow them will not be executed.

Defaults

If no options are specified, a limited initialization sequence is performed. The operating system and data communications are not affected but all of the emulation and analysis boards are reset. For example, a limited initialization would not change macro definitions, system date and time, or the data communications parameters, but the emulation memory map and breakpoint list would be reset to their default states.

Examples

To perform a powerup initialization sequence, type:

```
m> init -p
```

You will see:

```
                Copyright (c) Hewlett-Packard Co.  1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700 Series Emulation System
    Version:   A.00.00 20Nov87

  HP64742 Motorola 68000 emulator
  HP64740 Emulation Analyzer
```

To perform a complete initialization sequence, which resets the entire emulator without executing performance verification, type:

```
m> init -c
```

You will see:

```
                Copyright (c) Hewlett-Packard Co.  1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700 Series Emulation System
    Version:   A.00.00 20Nov87

  HP64742 Motorola 68000 emulator
  HP64740 Emulation Analyzer
```

To perform a limited initialization sequence, resetting only the emulator and analyzer, type:

```
m> init
```

You will see:

```
# Limited initialization completed
```

Related Commands

cf (change emulation configuration)

dt (set system date and time)

map (define the emulation memory map)

stty (set data communications parameters)

tinit (reset the analyzer to powerup defaults)

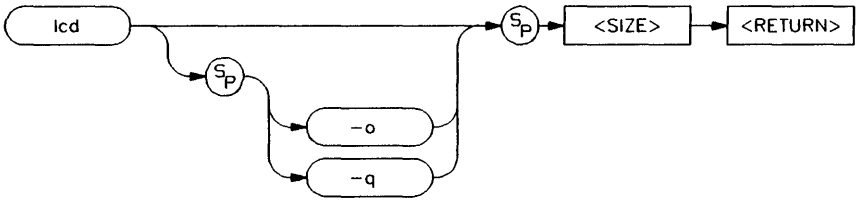
Notes

lcd

Summary

Load a block of emulator system control code (for use by high level interfaces to the HP 64700)

Syntax



Function

The **lcd** command allows a high level emulator software interface on a host system to load a block of executable system code into the HP 64700 emulator. This code might be used to add additional HP 64700 Terminal interface commands, modify commands that are in place, or setup macro commands. You will generally not use this command.

The code is downloaded in 68000 relocatable form on the currently active port, unless the **-o** option is specified.

Parameters

<SIZE>

<SIZE> specifies the size in bytes of the system code to be loaded.

-o

The **-o** parameter indicates that the code is to be downloaded from the **other** port (not the current command port).

When you use the **-o** option, the emulator enters the transparent mode. When the emulator is in the transparent mode, anything you enter from the keyboard is sent to the other port. Typically, you will enter

the command that downloads code into the emulator followed by the escape character sequence (see the **xp** command).

Once in the transparent mode, you must enter the escape character sequence before you can send characters to the emulator. For example, if you wish to abort the file transfer after you have entered the **lcd -o** command and are in transparent mode, you must enter the escape character sequence followed by a <CTRL>**c**.

-q

The **-q** parameter indicates that the operation should be performed in quiet mode; that is, the emulator system will not send "#" characters back to acknowledge the successful receipt of each record.

Defaults

If no options are specified, the code is expected at the current command port and a "#" character will be set for every record successfully processed. The code is downloaded in hex ASCII format.

Examples

To load a 292 byte block of code through the current command port in quiet mode, type:

```
M> lcd -q 292
```

Related Commands

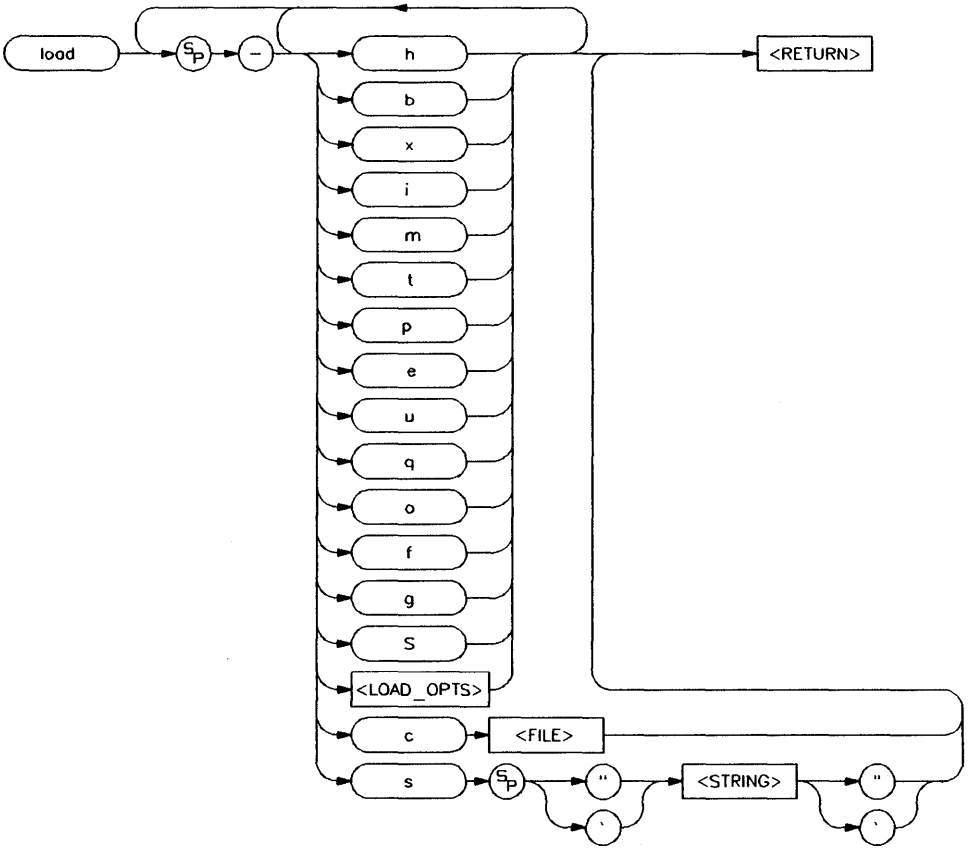
None.

load

Summary

Load user programs into emulation or target memory

Syntax



Function

The **load** command lets you load program code into emulation or target memory. Various file formats are supported via options to the load command. The destination of the program code is determined by the information contained in the program file.

Additional options allow you to load only target memory or emulation memory as desired.

If a load error occurs, the current load procedure is aborted. However, records which were successfully loaded will remain in memory.

For processors which use function codes, the function code information in the program file must conform to the specifications of the emulation memory mapper. For information on specifying emulator function codes, refer to the *Emulator User's Guide* for your particular emulator. You should also refer to the manuals supplied with your assembler or high-level language to determine how those tools specify function codes for your processor.

Parameters

Note

At least one dash (-) must be included before any parameters are specified. It is optional to include or omit dashes for succeeding parameters.

-i	Specifies that the program code will be in Intel hex file format.
-m	Specifies that the program code will be in Motorola S-record file format.
-t	Specifies that the program code will be in Tektronix hex file format.
-h	Specifies that the program code will be in HP file format. In this case, the file is expected to be transferred using the HP 64000 Hosted Development System **transfer** protocol.
-e	Load only those portions of program code which would reside in memory mapped to

emulation memory space. (Refer to the **map** command.)

-u

Load only those portions of program code which would reside in memory mapped to target memory space. (Refer to the **map** command.)

-q

The program code will be transferred in quiet mode. If **-q** is not specified, the emulator controller will write a "**#**" to **stdout** (as defined by the **po** command) for each record successfully received and processed.

-o

The program code will be received on the **other** port from the current command port.

When you use the **-o** option, the emulator enters the transparent mode. When the emulator is in the transparent mode, anything you enter from the keyboard is sent to the other port. Typically, you will enter the command that downloads code into the emulator followed by the escape character sequence (see the **xp** command).

Once in the transparent mode, you must enter the escape character sequence before you can send characters to the emulator. For example, if you wish to abort the file transfer after you have entered the **load -o** command and are in transparent mode, you must enter the escape character sequence followed by a <CTRL>**c**.

-s

If you supply the **-s** option along with a **<STRING>** parameter, the emulator transmits the given **<STRING>** out of the **other** port and expects to receive the program code there. The HP 64700 never enters transparent mode.

S	This allows you to download a symbol file from the host computer into the emulator. This option is valid for HP 64700 emulators that support the use of symbols.
<LOAD_OPTS>	This represents all options to the **load** command that are specific to a particular HP 64700-Series Emulator. Refer to your *Emulator Terminal Interface User's Guide* to see if your emulator supports any other **load** command options.
<FILE>	This represents the absolute file to load into the emulator.
<STRING>	**<STRING>**, along with the **-s** option, allows you to specify a string to be sent to a host computer attached to the other port. This string is intended to initiate the download process on the host. The string must be enclosed in single open quote marks or in double quote marks.

Note 

Many keyboards (and printers) represent the single open quote mark as an accent grave mark. In any case, the correct character is the character ASCII 60 hexadecimal. The correct double quote character is ASCII 22 hexadecimal.

-b	When using the HP file format, the program is expected to be in binary.
-x	When using the HP file format, the program is expected to be in hex.
-p	When using Intel, Motorola or Tektronix file formats, this option sets up a protocol checking scheme using ASCII **ACK/NAK**

characters. If using this option, the host should send one record at a time and wait for the emulator to return an ASCII **ACK** character between records. If the emulator returns an ASCII **NAK** instead, there has been an error in data transmission. When the emulator receives the EOF character, it will return only the normal emulator prompt since data transmission is complete.

If, during the transfer, the host receives a **NAK** for a record, it should retransmit the record until an **ACK** is received or until a timeout value is reached, whichever occurs first.

-c

If transferring from an HP 64000 station in terminal mode, this option requires that a filename be specified for the transfer. The emulator will pass the filename to the HP 64000 station.

If the **-c** option is specified, include the HP 64000 file name to be transferred in the form **<FILE_NAME>:<USERID>:absolute**.

-f

You specify the **-f** option if you are loading a foreground monitor into an HP 64700 emulator which supports a foreground monitor. Not all HP 64700 emulators which allow foreground monitors require this option for loading the monitor. Refer to the *Emulator User's Guide* for your particular emulator for further information on the use of this option.

-g

Certain emulators allow you to insert user code into the background monitor. This code directs the background monitor to perform certain functions unique to your target system. The **-g** option lets you load

this special code. Refer to the *Emulator User's Guide* for your particular emulator for further information on the use of this option.

Note

When you load an absolute file, the incoming data is examined for valid records (in the specified format). If the data being sent does not contain any valid records, the emulator will wait forever looking for valid records. The process must be terminated be entering a <CTRL>**c**.

Defaults

At least one file format option must be specified.

Examples

```
M> load -hbo<RETURN>
transfer -tb newfile.X<esc>*
```

Transfers a program file named newfile.X from an HP 9000 system running the transfer software to the other port of the HP 64700 emulator. The file will be transferred in HP absolute binary format.

You could accomplish the same transfer using the **load** command's **-s** option:

```
M> load -hbs "transfer -tb newfile.X"
```

Also, you could simplify repetitive loads using the **mac** command to define a macro named **ld**:

```
M> mac ld={load -hbs "transfer -tb
newfile.X"}
```

Now, to execute the load, just type:

```
M> ld
```

Now, let's look at an example that loads Motorola file formats:

```
M> load -mo
cat motosfile<esc>*
```

Transfers a program file named motosfile in Motorola S-record format to the other port of the HP 64700 emulator.

Note 

Both of these examples presume that the transparency escape character has been defined as a "*" by using the command **xp -f 01b -s 02a**.

If your emulator supports symbols, to load a symbol file from a host computer, for example, enter:

```
M> load -So
$ cat symbolfile <ESC>G xp -d
```

You could also load a symbol file using the **-s** (string) option with the **load** command. For example, you could enter:

```
M> load -Sos "cat symbolfile"
```

Related Commands

dump (allows you to transfer emulation memory contents to a host)

xp (controls transparent mode of emulator data communications ports)

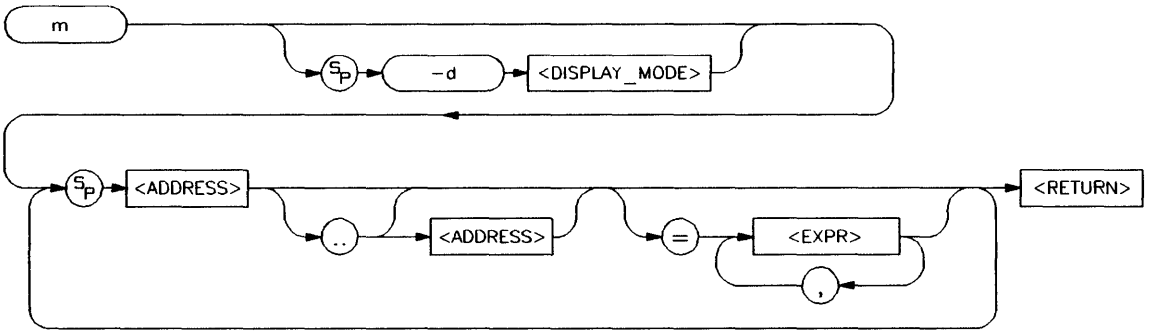
Notes

m

Summary

Display/modify memory locations

Syntax



Function

The **m** command allows you to display and modify emulation and target system memory. Options allow you to specify the display mode, specific address or addresses for display or modification, and the data values to be inserted.

If the selected address range for display or modification includes memory within the user's target system, the emulation processor will be broken to background upon execution of the command. After the command is complete, the processor will be returned to foreground execution if no errors occurred.

Note

The method of specifying address information varies among different types of microprocessors. Refer to the**address** syntax pages in the *Emulator User's Guide* for your particular emulator for specific address information. Remember that specifying an address a particular way in one command will affect the way you need to specify it for all commands. For example, if you use function codes in specifying a memory map, you will also need to use function codes within the address information for the **m** command to display or modify those ranges of memory.

Parameters

-d	The **-d** option allows you to set the display mode for memory accesses.
<DISPLAY_ MODE>	A one-character mnemonic specifying the display mode to use in creating memory displays. The allowable display modes are specific to the microprocessor in use; some typical modes are **b** (byte), **w** (word) and **m** (mnemonic). Refer to the **mode** syntax pages in the *Emulator User's Guide* for your emulator to determine the correct display modes. If no display mode is specified, the global display mode set via the **mo** command is used as a default.
<ADDRESS>	Specifies the address to be displayed or modified. As noted in the syntax, an address followed by two periods and another address specifies a range of addresses to display or modify. Address notation is specific to each microprocessor. For example, the 68000 emulator allows the use of function codes in specifying address information, whereas the Z80 emulator does not. However, for all processors the address default representation is a hexadecimal number.

Refer to the **<ADDRESS>** syntax pages in the *Emulator User's Guide* for your emulator for examples of correct address specifications.

Note 

If you specify only the first address of a range followed by two periods and omit the second address of the range, 128 bytes of the range starting at the first address specified are selected for display or modification.

<EXPR> Data value to which a particular location is to be modified. If a range of locations is to be modified to a sequence of data values, the values must be separated by commas. Refer to the examples for details.

Note 

The way the data item is handled depends on the <DISPLAY_MODE> in effect. For example, if the display mode is byte, and the data items 1a, 3f, and 66 are entered as 1a3f66, the location specified will be modified to 66 hex. If the display mode is word, the location will be modified to 3f66 hex. And if the display mode is long word, the location will be modified to 1a3f66. Note that data may be specified in decimal, octal, or binary in addition to the hexadecimal default. (Refer to the **<EXPR>** syntax pages for information on specifying numeric bases.) Conversely, if you specify the value 33 hex for modification in byte mode, the value 33 is entered; in word mode, the value 0033 is entered; in long word mode, the value 000033 is entered. In other words, if the value supplied is shorter than the mode in effect, it is padded with leading zeros.

Defaults At least one address must be specified. If no display mode is specified the display mode set by the **mo** command is used. Data items specified in memory modification are repeated as a group to

fill the address range specified (see the examples below for clarification). The memory <DISPLAY_MODE> defaults to the last value specified, or the default format for the emulator in use upon powerup initialization (varies dependent on the microprocessor being emulated).

Examples

Note

These examples were constructed using the 68000 emulator, but without the use of function codes. For information on using function codes or other microprocessor specific address specifiers, refer to the **<ADDRESS>** syntax pages in the *Emulator User's Guide* for your particular emulator.

To display the memory range 1000 hex through 101f hex in byte format, type:

```
M> m -db 1000..101f
```

You will see:

```
001000..00100f     00 00 30 00 00 00 40 00 54 48 49 53 20 49 53 20
001010..00101f     4d 45 53 53 41 47 45 20 41 54 48 49 20 49 53
```

To display the same address range in word format, type:

```
M> m -dw 1000..101f
```

You will see:

```
001000..00100f     0000 3000 0000 4000 5448 4953 2049 5320
001010..00101f     4d45 5353 4147 4520 4154 4849 5320 4953
```

To display the range in long word format (32 bits), type:

```
M> m -dl 1000..101f
```

You will see:

```
001000..00100f     00003000 00004000 54484953 20495320
001010..00101f     4d455353 41474520 41544849 53204953
```

For areas of memory that contain code, you may wish to display memory contents as assembler mnemonics. Type:

```
M> m -dm 2000..201f
```

You will see:

```
002000  2479000010  MOVEA.L 0001000,A2
002006  2679000010  MOVEA.L 0001004,A3
00200c  14bc0000    MOVE.B  #000,[A2]
002010  1012        MOVE.B  [A2],D0
002012  0c000000    CMPI.B  #000,D0
002016  67f8        BEQ.B   0002010
002018  0c000041    CMPI.B  #041,D0
00201c  6700000e    BEQ.W   000202C
```

Note

The instruction disassembler assumes that the first address location disassembled contains the first byte of an opcode; therefore, if you specify an address location that does not contain an opcode, the memory display will be incorrect.

Note

For the above examples, you should remember that the <DISPLAY_MODE> parameters vary depending on what modes are supported by your particular emulator. Refer to the **mode** syntax pages supplied with the *Emulator User's Guide* for your particular emulator for details on supported display modes for that emulator.

Remember that display modes default to the last one specified. Therefore, if you would like to examine data areas after using the mnemonic display mode, you should change the mode. Also, you can display more than a couple of rows of memory at a time. Type:

```
M> m -db 1000..10ff
```

You will see:

```
001000..00100f    00 00 30 00 00 00 40 00 54 48 49 53 20 49 53 20
001010..00101f    4d 45 53 53 41 47 45 20 41 54 48 49 53 20 49 53
001020..00102f    20 4d 45 53 53 41 47 45 20 42 49 4e 56 41 4c 49
001030..00103f    44 20 43 4f 4d 4d 41 4e 44 00 ff ef ff ff ff ff
001040..00104f    ff df bf ff 7f df ef ff ff dc 7f de ff df ff df
001050..00105f    ff cf ff df ff df ff df ff df ff df ff d6 ff df
001060..00106f    ff ff f7 ff 3f ff 7e f7 fe ff ff ff ff f7 f7 ff
001070..00107f    ff ff f7 ff ff ff ff ff ff 7f df ef ff ef f7 ef
001080..00108f    ff df e7 ff f7 df f7 ff ff df f6 df f7 bf f7 ff
001090..00109f    f7 df f7 ff f7 ff f7 ff 77 df f7 df ff df f7 bf
0010a0..0010af    ff ff f7 ff ff ff ff f7 ef ff ff ff ff fd fe f7
0010b0..0010bf    ff df ff ff ff e7 bf ff ff d9 ff df ff ff ff ff
0010c0..0010cf    f7 df ff ff f6 ff fe df f7 df f7 ff f7 dd f7 ff
0010d0..0010df    f7 df bf ff f7 ff f6 ff bf ff ff ff f7 f7 f6 ff
0010e0..0010ef    ff fe ff ff ff f7 ff ff ff df ff ff cb ff fd fe
0010f0..0010ff    ff f7 ff ff fd ff ff ff ff ff ff ff ff f7 fd df
```

Let's examine the memory modification capabilities. First, view the contents of location 5000 hex by typing:

```
M> m 5000
```

You will see:

```
005000..005000    41
```

Now modify the contents of 5000 hex to the byte value 21 hex by typing:

```
M> m 5000=21
```

Note

Notice that the results of the memory modification are not automatically displayed. To view the results of a modification, you enter another **m** command.

To view the new value of location 5000 hex, type:

```
M> m 5000
```

You will see:

```
005000..005000    21
```

Or you can clear the contents of a memory range. Type:

```
M> m 5000..501f=00
```

To display the results, type:

```
M> m 5000..501f
```

You will see:

```
005000..00500f     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
005010..00501f     00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
```

You can also modify the contents of a range to some other hex value. Type:

```
M> m 5000..501f=21
```

To view the results, type:

```
M> m 5000..501f
```

You will see:

```
005000..00500f     21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21
005010..00501f     21 21 21 21 21 21 21 21 21 21 21 21 21 21 21 21
```

A sequence of data items can be provided for modification. Type:

```
M> m 5000..501f=41,42,43
```

To view the results, type:

```
M> m 5000..501f
```

You will see:

```
005000..00500f     41 42 43 41 42 43 41 42 43 41 42 43 41 42 43 41
005010..00501f     42 43 41 42 43 41 42 43 41 42 43 41 42 43 41 42
```

Note

When a sequence of data items is provided for memory modification, the sequence is repeated until the entire range has been modified.

Microprocessors such as the 80186 which put most significant bytes in upper memory locations are handled correctly by the emulator. These examples were constructed on an 80186 emulator.

First, let's modify a block of memory to a range of incrementing values by typing:

```
R> m
700..7ff=0,1,2,3,4,5,6,7,8,9,0a,0b,0c,0d,0e,0f
```

Now, display that range in byte mode by typing:

```
R> m -db 700..
```

You will see:

```
00700..0070f  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00710..0071f  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00720..0072f  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00730..0073f  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00740..0074f  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00750..0075f  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00760..0076f  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
00770..0077f  00 01 02 03 04 05 06 07 08 09 0a 0b 0c 0d 0e 0f
```

Notice that the bytes were put in the proper positions. Now, display the memory range in word mode:

```
R> m -dw 700..
```

You will see:

```
00700..0070f  0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
00710..0071f  0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
00720..0072f  0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
00730..0073f  0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
00740..0074f  0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
00750..0075f  0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
00760..0076f  0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
00770..0077f  0100 0302 0504 0706 0908 0b0a 0d0c 0f0e
```

Notice that the bytes are swapped to represent significance. Now display in double word mode:

```
R> m -dd 700..
```

You will see:

```
00700..0070f  03020100 07060504 0b0a0908 0f0e0d0c
00710..0071f  03020100 07060504 0b0a0908 0f0e0d0c
00720..0072f  03020100 07060504 0b0a0908 0f0e0d0c
00730..0073f  03020100 07060504 0b0a0908 0f0e0d0c
00740..0074f  03020100 07060504 0b0a0908 0f0e0d0c
00750..0075f  03020100 07060504 0b0a0908 0f0e0d0c
00760..0076f  03020100 07060504 0b0a0908 0f0e0d0c
00770..0077f  03020100 07060504 0b0a0908 0f0e0d0c
```

Again, the bytes were reordered to represent their significance.

If your emulator supports symbols, you can display those symbols using the **memory display mnemonic** command. For example, using an 8051 emulator and an example program, you may enter:

```
R> m -dm CMD_RDR:INIT..FILL_DEST+5
```

You will see:

```
03000@x CMD_RDR:INIT      MOV SP,#fe
03003@x  MD_RDR:READ_CMD   MOVDPTR,#0000
03006@x  -                 MOV A,#00
03008@x  -                MOVX @DPTR,A
03009@x CMD_RDR:SCAN      MOVXA,@DPTR
0300a@x  -                 JZ CMD_RDR:SCAN
0300c@x  -                 CJNE A,#41,CMD_RDR:CMD_B
0300f@x  -                MOV R2,#12
03011@x  -                 MOV DPTR,#3100
03014@x  -                 SJMP CMD_RDR:WRITE_MSG
03016@x CMD_RDR:CMD_B     CJNE A,#42,CMD_RDR:CMD_I
03019@x  -                MOV R2,#12
0301b@x  -                 MOV DPTR,#3112
0301e@x  -                 SJMP CMD_RDR:WRITE_MSG
03020@x CMD_RDR:CMD_I     MOV R2,#10
03021@x  -                 JBC90,3055
03024@x  -                 ADD A,#ea
03026@x  -                XRL A,#ff
03028@x  -                 ADD A,#21
0302a@x  -                 MOV R3,A
0302b@x  -                INC DPS
0302d@x  -                 MOV DPTR,#0001
03030@x CMD_RDR:AGAIN     INC DPS
03032@x  -                 MOVXA,@DPTR
03033@x  -                 INC DPTR
03034@x  -                INC DPS
03036@x  -                 MOVX @DPTR,A
03037@x  -                 INC DPTR
03038@x  -                DJNZ R2,CMD_RDR:AGAIN
0303a@x  -                 MOV A,#00
0303c@x D_RDR:FILL_DEST   MOVX @DPTR,A
0303d@x  -                INC DPTR
0303e@x  -                 DJNZR3,CMD_RDR:FILL_DEST
03040@x  -                 INC DPS
03042@x  -                 SJMP CMD_RDR:READ_CMD
```

Note

The command processor retains the name of the last module referenced. If a symbol does not contain a module name, the list of global symbols is searched. If the symbol is not found, the list of user symbols is searched. If the symbol is still not found, the system searches the last module referenced. If it doesn't find it there, the rest of the modules are searched.

Related Commands

map (specify mapping of memory to emulation or user memory and to RAM or ROM)

mo (specify global access and display modes)

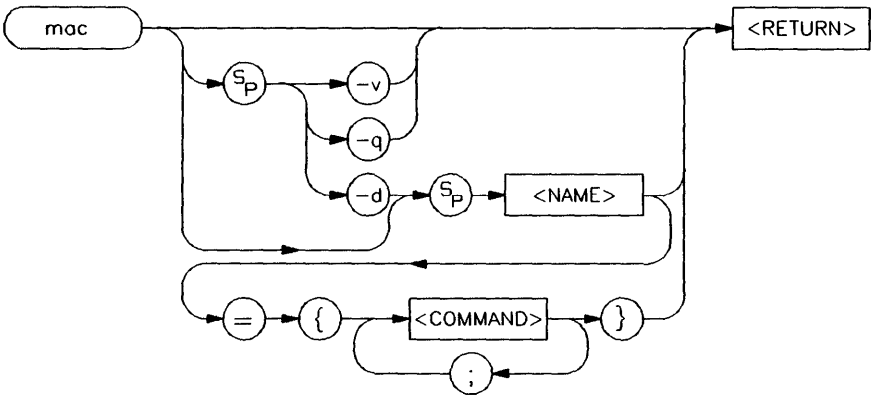
io (display modify I/O locations (for processors which support dedicated I/O))

mac

Summary

Define command macros

Syntax



Function

The **mac** command allows you to save a group of commands under a name of your choice. This allows you to instantly recall that command group by typing in the assigned name; the emulator will then preprocess the macro to expand the commands stored therein to a normal command line; the command line is then executed as usual.

Nested macro calls are permitted and limited only by constraints of system memory.

The commands within the macro definition are not checked for correct syntax until the macro is executed; therefore, it is advisable to test the command string before defining the macro.

The number of macros that can be created is limited to 100, but may be less depending on the complexity of the macros defined.

The length of the macro name combined with the macro definition is limited only by the maximum HP 64700 command length of 255 characters; thus, the macro name and definition can be a maximum of 251 characters.

A command within a macro definition cannot contain the pound sign character (#) unless the command is enclosed in a quoted string. (Otherwise, text following the # is interpreted as a comment.) This means there can be no matching brace at the end of the command. Use the **echo** command to place comments in a macro definition.

Command line substitution is possible when invoking a macro. During the macro definition, you may include pseudo-parameters which allow you to substitute parameters, such as file names, when invoking the macro.

Parameters

-d	The **-d** parameter, in conjunction with the macro <NAME>, deletes the macro defined by <NAME>. If <NAME> is given as the character "*" then all macros are deleted.
<NAME>	This represents the name you assign to the macro definition. Names can be any combination of alphanumeric characters; however, you cannot define a macro that has a name identical to that of another HP 64700 Terminal Interface command. If you specify a name which is the same as a currently defined macro, that macro will be overwritten by the new macro you define.

Note 

Certain HP 64700-Series emulators may predefine macros to aid you in setting up configurations for certain emulation tasks, such as in-circuit emulation.

<COMMAND> This represents one or more emulator commands, including names which are used to define other macros. <NAME> and <COMMAND> must be separated by an equal sign (=), and the command string must be enclosed with braces "{ }." Each <COMMAND> must be separated from other commands by a semicolon (;).

When using command substitution, you can include pseudo-parameters in the form of "&token&" in the macro definition. Do not include any white space between the two "&" symbols. When you execute the macro, include the string to be substituted for &token& as a parameter on the command line. The macro will execute using the command expanded with the string you substituted. See the Examples section for more information.

-q This option sets the macro expansion echo to quiet mode. In this mode, any macro that you run will be executed without displaying the expanded command string.

-v This option sets the macro expansion echo to verbose mode. In this mode, any macro that you run will first display the expanded command string as a comment, and then will execute the macro.

Defaults

If no parameters are supplied, the current set of macro definitions is displayed. If only <NAME> is supplied without a command string, the macro defined by <NAME> is displayed.

Examples

Let's define a macro that resets the emulator, then defines the memory map, resets the processor and breaks into the monitor, then sets up the stack pointer. Type:

```
M> mac setup={init;map 0..7fff eram;rst
-m;reg usp=7000}
```

To execute the command, type:

```
M> setup
```

You will see:

```
# init ; map 0..7fff eram ; rst -m ; reg usp=7000
     All products re-initialized
```

To illustrate the changes caused by each command, let's look at the memory map and registers. Type:

```
M> map
```

You will see:

```
# remaining number of terms    : 6
# remaining emulation memory   : 17800h bytes
map  000000..007fff     eram    # term 1
map  other tram
```

Type:

```
M> reg
```

You will see:

```
reg pc=00000000 st=0000 d0=00000000 d1=00000000 d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00000000 a1=00000000
reg a2=00000000 a3=00000000 a4=00000000 a5=00000000 a6=00000000 a7=00007000
reg usp=00007000 ssp=00000006
```

You could define another macro called "echonwait" as follows:

```
M> mac echonwait={echo "Set S1 to OFF";w}
```

To see what macros are currently defined, type:

```
M> mac
```

You will see:

```
mac  setup={init ; map 0..7fff eram ; rst -m ; reg usp=7000 }
mac  echonwait={echo "Set S1 to OFF" ; w}
```

To delete the macro named **setup** that we just defined, type:

```
M> mac -d setup
```

Verify that the macro named **setup** was in fact deleted by typing:

```
M> mac
```

You will see:

```
mac  echonwait={echo "Set S1 to OFF" ; w}
```

To delete the remaining macro, you can either specify it by name, or simply type:

```
M> mac -d *
```

Now type:

```
M> mac
```

Since no macros remain, you will see:

```
M>
```

To define a macro with a pseudo-parameter that allows you to substitute a file name, enter:

```
M> mac getfile={load -hbs"transfer -t
&file&"}
```

To use the macro with the pseudo-parameter, and substitute the pseudo-parameter (&file&) with an example file name, enter:

```
M> getfile YOURFILE.o
```

If the macro expansion mode is set to verbose (**-v** option is used), you will see the macro and associated parameters expand to include your file name. For example, you will see:

```
# load -hbs "transfer -t YOURFILE.o"
```

Pseudo-parameters are replaced on a position-dependent scheme, where the first pseudo-parameter encountered in the macro string is replaced with the first parameter passed into the macro. The second pseudo-parameter is replaced with the second parameter passed into the macro, and so on.

As an example of multiple parameter substitution, let's define a macro that fills an arbitrary 100-byte block range with a user-defined value. To do this, enter:

```
M> mac fill={equ start=&address&;m -db
start..start+100t=&value&}
```

To invoke the macro, enter:

```
M> fill  50  88
```

In this example, 50 will be substituted for &address&, and 88 will be substituted for &value&. So, addresses 50 through 150 decimal will contain the value 88.

Note 

You can define multiple pseudo-parameters in a macro using the same name for both (or all) of them. Because pseudo-parameters are position-dependent, the first pseudo-parameter will always be substituted with the first parameter you pass into the macro, the second pseudo-parameter with the second parameter you pass into the macro, and so on.

Related Commands

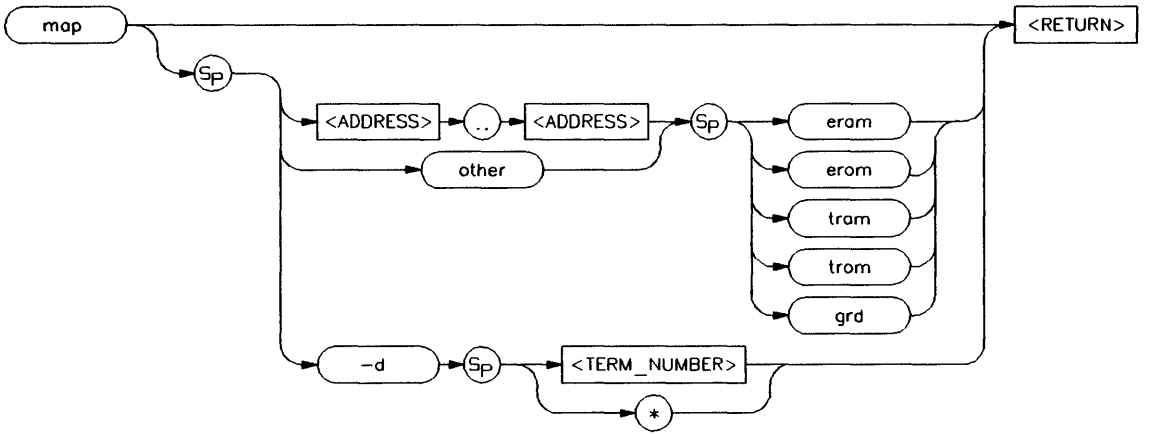
rep (repeat; allows you to repeat any command, including macros)

map

Summary

Map emulation and target system memory

Syntax



Function

The **map** command allows you to map address ranges to one of five different classes of memory. For example, you may want to specify that addresses 1000 through 2fff hex are in emulation RAM, and addresses 3000 through 3fff hex (where your program code will reside) are in emulation ROM. Later, when your target system hardware is prototyped, you will be able to easily modify these specifications to indicate that the address ranges actually reside in target system RAM or ROM.

The emulation system assigns a term number to each address range specified by you in the map command. Term numbers are assigned in ascending order of address range. Therefore, if you map the addresses 0 through 100 (TERM_NUMBER_1) and 1000 through 1fff (TERM_NUMBER_2), then specify another range of 300 through 3ff, TERM_NUMBER_2 will be renumbered as TERM_NUMBER_3 and the range 300 through 3ff will become TERM_NUMBER_2. Remember to use the assigned term

number when specifying mapper terms to be deleted by the **map -d <TERM_NUMBER>** command.

Note

The memory mapper re-assigns blocks of emulation memory after the insertion or deletion of mapper terms. For example, if you modified the contents of 300 through 3ff above, deleted TERM_NUMBER_1, and displayed locations 300 through 3ff, you would notice the contents of those locations are not the same as they were before deleting the mapper term.

Mapper address block sizes vary with each individual emulator. However, the block sizes for target memory and emulation memory on a particular emulator are identical. If an address range smaller than a multiple of the block size is entered as a map specification, the range is rounded upwards to the nearest block size multiple.

When any map term is added or deleted the emulation processor will be reset and held in the reset state until a break or run command is issued.

Note

The processor remains reset in recognition of the fact that returning to execution directly after mapper modification is most likely invalid.

Note

Be sure to disable all breakpoints (**bc -d bp**) before changing the map. Breakpoints are not cleared when the memory map is changed. (Breakpoints are also not cleared when a file is loaded, or when memory is manually modified.) After the new map and the program are set up, you can re-enable the breakpoints by re-enabling the breakpoints break condition (**bc -e bp**) and entering the **bp -e *** command. When the list of breakpoints is displayed (**bp**), the memory is checked to verify whether the breakpoint is still in memory.

Each type of emulator has its own default memory map at powerup. If all mapper terms are deleted with the command **map -d ***, the "other" range is unaffected. The number of map terms available depends on the emulator in use; for example, the HP 64700 emulator for the 68000 has 7 map terms available while the emulator for the Z80 has 16 map terms available. Refer to the *Emulator User's Guide* for details on the maximum number of map terms.

Parameters

<ADDRESS>

The address values specify the address range to be assigned to a particular memory type. Whenever the emulation processor accesses the range specified, it will be directed to the memory type specified in the map. Specification of address information defaults to a hexadecimal value; some HP 64700 emulators, such as that for the 68000, allow specification of additional address specifiers such as function codes. Refer to the **<ADDRESS>** syntax pages in the *Emulator User's Guide* for your emulator for details of address specification.

other

The address range **other** specifies all address ranges not otherwise specified by mapper terms. Certain HP 64700 emulators restrict

type definition of the "other" range to **trom**, **tram**, or **grd**.

eram

Specifying **eram** indicates that the given address range is to reside in emulation address space and act as RAM (read/write).

erom

Specifying **erom** indicates that the given address range resides in emulation address space; it is to act as ROM (read only). The **bc** command allows you to specify that emulation processor writes to this space or to space designated as target ROM (**trom**) will cause an emulation system break.

Current HP 64700 Emulators protect emulation memory from being modified when a write to emulation ROM occurs. (This feature may not be supported in future HP 64700-Series emulators.)

tram

Specifying **tram** indicates that the given address range lies within target system RAM space. When the emulation processor accesses an address within this range, the target system data buffers will be enabled by a mapper signal to complete the transaction.

trom

Specifying **trom** indicates that the given address range lies within target system ROM space. As with the **erom** parameter above, the **bc** command may be used to set up the emulation system to break upon a write to these address ranges. In any case, if target ROM memory is actually implemented as RAM, and the necessary write strobes are connected to this memory, the emulator will allow the processor to overwrite the memory locations.

grd — The **grd** parameter indicates the given address range is to be "guarded"; therefore, the emulation system software should not know that it exists. An emulation system break will always be generated upon accesses to guarded memory.

Defaults

If the command **map** is entered with no parameters, the current memory map is displayed.

Memory maps for each emulator type may differ. Refer to the *Emulator User's Guide* for your emulator for details.

Examples

These examples were created with a HP 64700 68000 emulator, without the use of function codes. The 68000 uses a default block size of 512 bytes for both user and target system memory.

To view the power up memory map, type:

```
M> map
```

You will see:

```
# remaining number of terms    : 7
# remaining emulation memory   : 1f800h bytes
map  other tram
```

The first line lists the number of terms available for mapping (in this case seven), the second line lists the number of bytes available in emulation memory, and the last line lists the only map term. In this case, all emulation memory is mapped as belonging to target system RAM. You will notice in the examples below that the other term doesn't have a term number. (The **other** term does not occupy a term number.)

Now let's map some address ranges to various types of memory. Type:

```
M> map 0..1ff eram
R> map 0..3ff erom (notice that the emulator
is now reset)
R> map 1000..15ff tram
R> map 2000..201f trom
```

You can view the resulting changes by typing:

```
R> map
```

You will see:

```
# remaining number of terms   : 3
# remaining emulation memory  : 1f400h bytes
map  000000..0001ff     eram    # term 1
map  000200..0003ff     erom    # term 2
map  001000..0015ff     tram    # term 3
map  002000..0021ff     trom    # term 4
map  other tram
```

Note that term 4 has a greater address range than what you originally specified (2000..21ff instead of 2000..201f). This is because the 68000 emulator defaults to map terms with a multiple of 512 byte blocks. The system therefore rounded your map entry up to the nearest 512 byte boundary while creating the map term.

You can map all other memory to "guarded"; the emulator will generate a break if a guarded memory access occurs, notifying you that something is probably awry with your program. Type:

```
R> map other grd
```

View the changes in the map:

```
R> map
```

You will see:

```
# remaining number of terms   : 2
# remaining emulation memory  : 1f400h bytes
map  000000..0001ff     eram    # term 1
map  000200..0003ff     erom    # term 2
map  001000..0015ff     tram    # term 3
map  002000..0021ff     trom    # term 4
map  003000..0031ff     trom    # term 5
map  other  grd
```

Maybe you decided term 4 really wasn't what you wanted. Type:

```
R> map -d 4
```

View the changes:

```
R> map
```

You will see:

```
# remaining number of terms    : 3
# remaining emulation memory   : 1f400h bytes
map  000000..0001ff      eram    # term 1
map  000200..0003ff      erom    # term 2
map  001000..0015ff      tram    # term 3
map  003000..0031ff      trom    # term 4
map  other  grd
```

Instead, what you really needed was an emulation ROM term from 400 through 5ff hex. Type:

```
R> map 400..5ff erom
```

You will see:

```
# remaining number of terms    : 2
# remaining emulation memory   : 1f400h bytes
map  000000..0001ff      eram    # term 1
map  000200..0003ff      erom    # term 2
map  000400..0005ff      erom    # term 3
map  001000..0015ff      tram    # term 4
map  003000..0031ff      trom    # term 5
map  other  grd
```

Notice that term 3 and term 4 were renumbered as term 4 and term 5; the new entry was inserted as term 3. Mapper terms are arranged in ascending address order.

To delete all of the map terms (reset the map), type:

```
R> map -d *
```

If you now type **map**, you'll see the same display from the first example.

Related Commands

bc (break conditions; determines whether emulator breaks to monitor upon write to space mapped as ROM)

m (memory display/modify)

bp (set/delete software breakpoints)

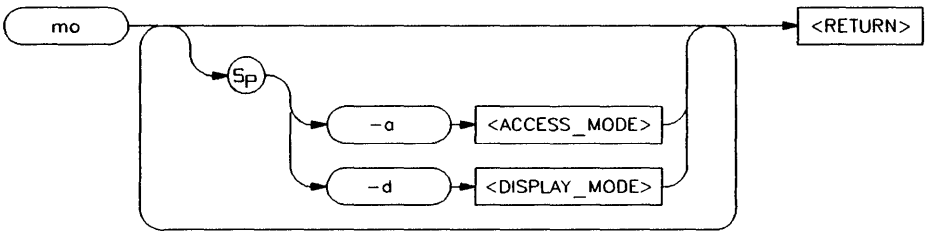
Notes

mo

Summary

Set global memory access and display modes

Syntax



Function

The **mo** command allows you to modify the global access and display modes. Access mode is defined as the type of processor data cycles used by the emulation monitor to access a portion of user memory. Display mode is defined as the method used to display or modify data resident in memory.

The options for the access and display mode vary for each microprocessor type, as each processor supports different data types for memory transactions. Refer to the *Emulator User's Guide* for your particular emulator for details on the supported modes.

Parameters

-a	The **-a** parameter in combination with a single character specifying mode type sets the global access mode.
<ACCESS_ MODE>	A single character used to specify the global access mode. Note that there is no space between the **-a** parameter and the mode specifier. Typical mode types are **b** (byte) and **w** (word), although the types supported are dependent on the microprocessor in use.

For further information, see the syntax pages for <MODE> in the *Emulator User's Guide* for your particular emulator.

-d — The **-d** parameter in combination with a single character sets the global display mode default.

<DISPLAY_MODE> — A single character used to specify the global display mode default. Note that there is no space between the **-d** parameter and the mode specifier. Typical mode types are **b** (byte), **w** (word), **l** (long word), and **m** (mnemonic); however, the types available are dependent on the microprocessor in use. For further information, see the syntax pages for <MODE> in the *Emulator User's Guide* for your particular emulator.

Defaults

If no parameters are specified, the current settings of the display and access modes are displayed.

Examples

The examples below were created on a Motorola 68000 emulator.

Note

For examples of the effects of changing the display mode, refer to the syntax pages for the **m** (memory) command in this manual.

To view the current settings of the access and display modes, type:

```
R> mo
```

You will see:

```
mo -ab -db
```

Now, to set the access mode to words, type:

```
R> mo -aw
```

No response is returned when a mode setting is changed. To verify that the mode status did in fact change, type:

```
R> mo
```

You will see:

```
mo -aw -db
```

To change the access mode to words and the display mode to long words, type:

```
R> mo -aw -dl
```

To verify that the mode has changed, type:

```
R> mo
```

You will see:

```
mo -aw -dl
```

To change the access mode to words, and the display mode to mnemonics, type:

```
R> mo -aw -dm
```

To reset the access and display modes to the powerup defaults, type:

```
R> mo -ab -db
```

Related Commands

m (memory display/modify)

io (input/output display or modify)

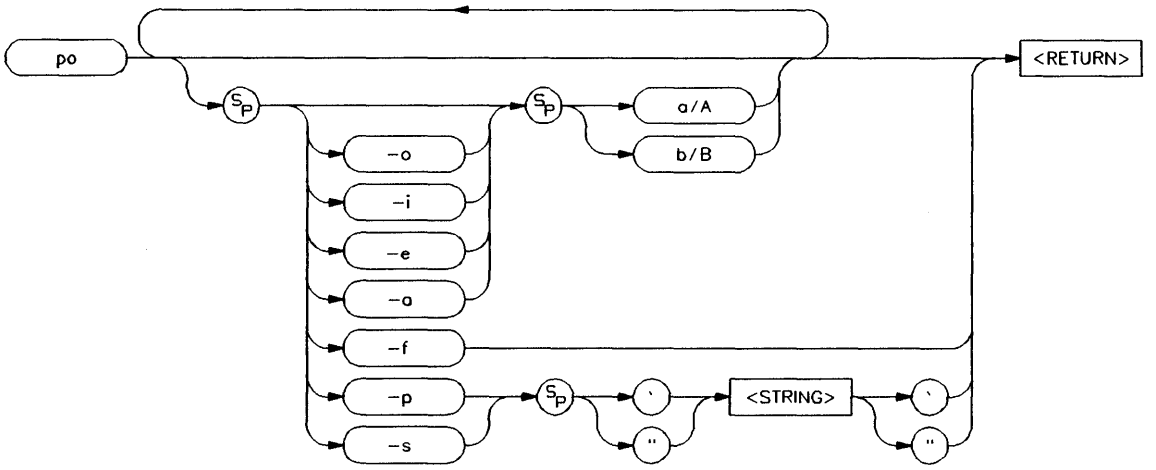
Notes

po

Summary

Assign ports, redefine prompt, dump command files

Syntax



Function

The **po** command allows you to change the port used for standard input, output, and error. Either Port A (Main) or Port B may have any or all of the above device associations (but not both ports with the same device association).

Other options provided with the **po** command allow you to inform the emulator that a command file is to be read from the "other" port (not the current command port) or change the system prompt characters.

When using the command file option, one port of the emulator is hooked to a controlling terminal, the other port is hooked to a host computer (transparent configuration). When the **po -f** command is given, the emulator immediately enters transparent mode, allowing you to issue a command to the host which will send the desired command file to the HP 64700 emulator. The **po -s "<STRING>"**

command is the same as the **po -f** command except you specify the host command in the string.

Parameters

-o — The **-o** option indicates that the standard output device (abbreviated as stdout) is to be attached to the specified port.

Note

The command prompt will be returned to the currently designated input port; however, all other output, such as the results of memory display commands, will be directed to the newly designated output port.

-i — The **-i** option indicates that the standard input device (abbreviated as stdin) is to be attached to the specified port.

Note

Once the **po -i <PORT>** command is issued, all further command input must come from the designated <PORT>. Therefore, be certain you have a way of supplying command input to the designated port; otherwise, you will have to enter **<CTRL>-C** to regain control of the emulator's command port. As a last resort, cycle the HP 64700 emulator power switch to reset to the default configuration and regain control of the emulator.

-e — The **-e** option indicates that the standard error list device (abbreviated as stderr) is to be attached to the specified port.

-a — The **-a** option indicates that all the devices, including standard input, output, and error, are to be attached to the specified port.

-f	The **-f** parameter indicates that the emulator should read a command file from the other port (the one not designated as **stdin**). This allows you to save sets of HP 64700 commands on a host computer, then reload them as necessary.
-p	The **-p** option allows you to change the emulator's command prompt to one specified by **<STRING>**.
<STRING>	**<STRING>** is any group of ASCII characters enclosed by single open quotes (') or double (") quote marks. This parameter, when used with **-p**, allows you to specify a new emulator command prompt.
-s	If you supply the **-s** option along with a **<STRING>** parameter, the emulator transmits the given **<STRING>** out of the other port and expects to receive the command file there. The string is the host command that transfers the command file to the HP 64700. The HP 64700 never enters transparent mode.
A	The devices operated on are to be attached to Port A (Main).
B	The devices operated on are to be attached to Port B.

Note 

Port A (Main) and Port B correspond to the two identically named DB-25 pin connectors on the rear panel of the HP 64700-Series emulator.

Defaults

If no parameters are specified, the current configurations of stdin, stdout, and stderr are displayed.

Examples

To display the current status of stdin, stdout, and stderr, type:

```
M> po
```

You will see:

```
po -o A
po -e A
po -i A
```

To attach the standard output device (stdout) to port B, type:

```
M> po -o B
```

To attach stderr to Port B and stdout to Port A, type:

```
M> po -e B -o A
```

Now verify the current port assignments by typing:

```
M> po
```

You will see:

```
po -o A
po -e B
po -i A
```

Assign both the standard input and output devices to Port B:

```
M> po -o B -i B
```

After execution of this command, the original controlling device on Port A will no longer communicate with the emulator. You must have a means for issuing commands to the emulator from the device connected to Port B; otherwise, you will need to enter the key combination **<CTRL>-C** to regain control of the emulator's command port. You could also cycle power to restore the default port assignments set by the emulator configuration switches (although that is a somewhat more drastic solution). (Refer to the *Hardware Installation and Configuration* manual for further information on the emulator configuration switches.)

To view the new assignments, issue the following command from Port B:

```
M> po
```

The following information will be returned to Port B:

```
po -o B
po -e B
po -i B
```

To reset all the device assignments to Port A, issue the following command from Port B:

```
M> po -a A
```

Control of the emulator is now returned to the terminal (or host computer) connected to Port A.

If you are using the emulator in a transparent configuration, with a terminal on Port A and a host computer (such as an HP 9000/300 system running HP-UX) on Port B, you may wish to define command files which would set up the emulator whenever you need to start an emulation session.

For example, you might want to load the memory map, change the emulator configuration items, then load equates and macros. You can use the **po** command along with the command macro capability to accomplish these things.

First, you must define a macro to save all of the current emulator configuration items. (These are the items that you will read back in later as commands.) Type:

```
M> mac sconfig={stty b ocr;po -o b;echo "cat
cfile";w 1;map;cf;equ;mac;echo "#+#";echo
\04;po -o a}
```

The HP 64700 emulator will react as follows when you execute this macro:

stty b ocr -- sets up port B to terminate each text string with a carriage return on output. (The default is new line & carriage return; using this method ensures that there are no gaps in your command file when saved.)

po -o b -- defines port B as **stdout**.

echo "cat cfile" -- Since **stdout** is now to port B, which is connected to the host computer, this echoes a command to the host that directs all further information from port B into a host file named **cfile.**

w 1 -- a one second wait which allows the host enough time to open the file named **cfile** and redirect host input (output from HP 64700 port B) to the file.

map -- writes the current emulator memory map to the file named **cfile.**

cf -- writes the current emulator configuration (from the **cf** command) to the file.

equ -- writes the currently defined equate statements to the file.

mac -- writes all of the currently defined macros to the file. Note that since you have defined the macro **sconfig,** it will also be written to the file. You can later reuse this macro to save changes in the emulator configuration.

echo "#-#" -- this will be the last line in **cfile.** When the file is sent to the emulator at a later time, it will signal the end of command file processing.

echo \04 -- sends a <CTRL>-D (EOF) character to close **cfile.**

po -o a -- resets the **stdout** device to port A; further commands will send their output to port A.

Now the file named **cfile** is on your host computer and can be used to reconfigure your emulator as follows:

```
M> po -f
cat cfile <ESC><CHAR>
```

The **po -f** command indicates that the emulator should go into transparent mode and process a command file to be received on the "other" port (not the current command port). You then enter the **cat cfile** command which instructs the host to write the contents of **cfile** to its **stdout** port (which is connected to the "other" port of the HP 64700). Do NOT press <RETURN> after entering the command. Instead, you should enter the two character transparent mode escape sequence (as defined by the **xp**

command). A carriage return will be sent to the host by the emulator when it has finished processing the command file.

A command equivalent to the **po -f** command above is:

```
M> po -s "cat cfile"
```

Note that the emulator will read the contents of the file **cfile** it is receiving and execute these as commands until the "#+#" characters are received.

Note

If the command file does not contain the end-of-command string "#+#", it will still return to the normal operating mode if no more commands are read within 10 seconds.

You can redefine the emulator's command prompt string using the **po -p <STRING>** command. Upon powerup, the emulator prompt defaults to "". (The character before the string, for example, **R**, **M**, **U**, etc., is used to indicate the current emulator status and is NOT affected by redefining the prompt string.)

For example, you may want to redefine the prompt string to your name so that others who use the emulator will know you are currently using the system.

The standard prompt is:

```
U>
```

To redefine it with a separator character plus your name and a prompt, type:

```
U> po -p "\YOURID>"
```

You will see:

```
U\YOURID>
```

To redefine the prompt back to the original, type:

```
U\YOURID> po -p ">"
```

You will see:

```
U>
```

If several people use the system, you may want to define macros which reset the prompt so each user knows who is currently using the emulator. For example:

```
M> mac yourid={po -p "\YOURID>"}
M> mac herid={po -p "\HERID>"}
M> mac hisid={po -p "\HISID>"}
```

Related Commands

stty (set up data communications parameters for ports)

xp (set up and enable/disable transparent mode)

Note

The default port used upon powerup initialization for stdin, stdout, and stderr can be changed using the emulator configuration switches. Refer to the *HP 64700 Hardware Installation and Configuration* manual for further information.

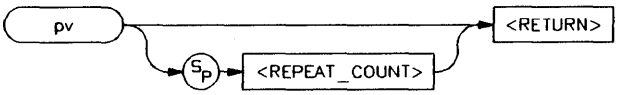
---

pv

Summary

Run emulator/analyzer performance verification

Syntax



Function

The **pv** command runs performance verification on the emulator and analyzer. The performance verification exercises all the emulator hardware and software to high confidence level.

You should only run performance verification when the emulation probe is **not** plugged into a target system. You should also make sure to remove any conductive foam or plastic pin protectors from the emulator probe, as these will cause failures during performance verification.

Note

When you use the **pv** command, the emulator is initialized as if power were cycled. Therefore, all equates, macros, memory map, configuration settings, system clock, software breakpoints, trace specifications, and other configuration items you have altered will be cleared. Do not use the **pv** command unless you can restore these items from a host, or have documented them so you can restore their states manually.

If **pv** reports failures, first check your hardware installation as documented in the *Hardware Installation and Configuration* manual. If the failures persist, call your local HP Sales and Service office for assistance. A list of offices is provided in the *Support Services* guide.

Note that providing multiple commands such as **pv 1;r** is invalid; the second command will not execute due to the system reset.

Typing in <CTRL>-C to abort the **pv** command may result in incorrect failure messages.

Parameters

<REPEAT_ COUNT>	**<REPEAT_COUNT>** allows you to specify the number of times to repeat the performance verification. This is a required parameter.

Defaults

If no parameters are given, a warning message about initialization of the emulator along with correct **pv** command syntax is displayed. To actually execute the **pv** command, you must provide a <REPEAT_COUNT> value.

Examples

Executing **pv** with no parameters provides a warning display, along with help for the correct syntax. Type:

```
M> pv
```

```
***** WARNING *****

   Running this pv (Performance Verification) command destroys
the current system configuration by performing a cold system reboot
after the command has completed.  All system and emulation setups
including the date, macros, equates, memory map, and configuration
items will be returned to their default powerup states.

   The emulation probe should be disconnected from the target
system before running this performance verification.

   To run pv type:   "pv <repeat_count>"
```

To loop through the performance verification twice, type:

```
M> pv 2
```

The example shown below is for an HP 68000 emulator.

```
  Testing: HP64742 Motorola 68000 emulator
    PASSED
    Number of tests: 1           Number of failures: 0
  Testing: HP64740 Emulation Analyzer
    PASSED
    Number of tests: 1           Number of failures: 0

  Testing: HP64742 Motorola 68000 emulator
    PASSED
    Number of tests: 2           Number of failures: 0
  Testing: HP64740 Emulation Analyzer
    PASSED
    Number of tests: 2           Number of failures: 0

                Copyright (c) Hewlett-Packard Co.  1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700 Series Emulation System
    Version:   A.00.01 03Dec87 Unreleased

  HP64742 Motorola 68000 emulator
  HP64740 Emulation Analyzer
R>
```

Notice that the emulator initializes and returns to the reset state.

Related Commands

init (reinitializes the emulator)

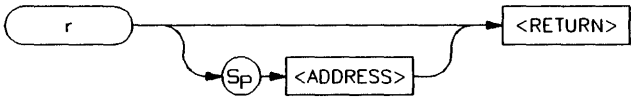
Notes

r

Summary

Run the emulator from current PC or specified location

Syntax



Function

The **r** command starts an emulation run. Execution begins at the address specified by the <ADDRESS> parameter; if no address is specified, execution begins at the address currently present in the program counter.

Parameters

<ADDRESS>	Specifies the address where execution is to begin. If you specify **$**, the processor runs from the current program counter value. If you specify **rst**, the processor runs from its reset address.

Note

Different microprocessors have different addressing capabilities and so do the emulators. Although <ADDRESS> defaults to a hexadecimal number, each processor may allow specification of additional address information (for example, the 68000 function codes). Refer to the <ADDRESS> syntax pages in the *Emulator User's Guide* for your particular emulator.

Note

Each emulator behaves differently on receiving a **r rst** command. Refer to the *Emulator User's Guide* for further information.

Defaults

If no parameters are specified, the emulation run begins at the address specified by the processor's current program counter contents.

Examples

To start your program running from address 2000 hex, type:

```
R> r 2000
U>
```

(Note that the prompt changes to indicate the emulator is running user code.)

Now let's issue a break command. Type:

```
U> b
M>
```

(Note that the prompt changes to indicate the emulator is running in the emulation monitor.)

Let's view the current program counter value. Type:

```
M> reg
```

You will see:

```
reg pc=00002010 st=2004 d0=00000000 d1=00000000 d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00000000 a1=00000000
reg a2=00003000 a3=00004000 a4=00000000 a5=00000000 a6=00000000 a7=00000006
reg usp=00007000 ssp=00000006
```

To run the emulator from the current program counter value of 2012 hex, type:

```
M> r
U>
```

(Note the prompt changes again to indicate the emulator is running in user code.)

Related Commands

s (step; allows controlled stepping through program instructions)

rx (run only when CMB (Coordinated Measurement Bus) execute pulse is received)

x (pulse the CMB execute line if resident on the CMB)

Refer to the *CMB User's Guide* for information on the **r** command's effect on the CMB.

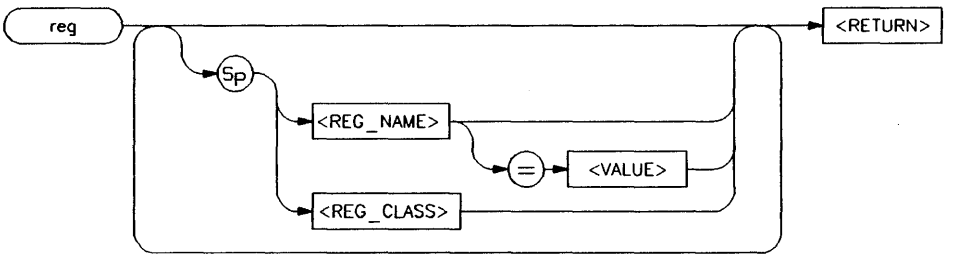
Notes

reg

Summary

Display/modify processor registers

Syntax



Function

The **reg** command allows you to display and modify emulation processor register contents. Individual registers may be displayed or modified; related groups of registers may be displayed; combinations of display and modify are permitted on the same command line.

Parameters

<REG_NAME> The <REG_NAME> parameter allows you to specify a specific register to display or modify. The valid register names are microprocessor dependent; therefore, refer to the <REGISTERS> syntax pages in the *Emulator User's Guide* for your emulator for the list of valid register names.

<REG_CLASS> The <REG_CLASS> parameter allows you to specify an entire group of registers for display. The group names vary from processor to processor; therefore, refer to the <REGISTERS> syntax pages in the

Emulator User's Guide for your emulator for a list of the valid register classes.

<VALUE> To modify a register's contents, supply the new contents in the <VALUE> variable. This is a numeric value (default is hexadecimal, other number bases may be specified.)

Examples

These examples were constructed using the HP 64700 Emulator for the 68000 microprocessor and the 68000 sample program listed in Appendix A.

To view the register values before the program is run, type:

```
M> reg
```

You will see:

```
reg pc=00000000 st=0000 d0=00000000 d1=00000000 d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00000000 a1=00000000
reg a2=00000000 a3=00000000 a4=00000000 a5=00000000 a6=00000000 a7=00007000
reg usp=00007000 ssp=00000006
```

Now let's see how the register contents are affected by executing only the MOVE.L INPUT_POINTER,A2 instruction which begins the program at address 2000 hex. To do this, we will use the **s** (step) command. Type:

```
M> s 1 2000
```

You will see:

```
002000  2479000010  MOVEA.L 0001000,A2
PC = 002006@sp
```

View the new register values by typing:

M> **reg**

You will see:

```
reg pc=00002006 st=2000 d0=00000000 d1=00000000 d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00000000 a1=00000000
reg a2=00003000 a3=00000000 a4=00000000 a5=00000000 a6=00000000 a7=00000006
reg usp=00007000 ssp=00000006
```

Notice that the following registers have changed: The program counter **pc** now points to the next instruction; the status register **st** bit 13 is set to indicate that the processor is running in the supervisor state (since we did not set up the state, the default is supervisor state); and address register **a2** contains the value 3000 hex as a result of the move instruction (long word contents of address 1000 hex were moved to **a2**).

Step the program again from the current program counter value:

```
M> s 1
```

You will see:

```
002006@sp  2679000010  MOVEA.L 0001004,A3
PC = 00200c@sp
```

Again, view the register contents:

```
M> reg
```

You will see:

```
reg pc=0000200c st=2000 d0=00000000 d1=00000000 d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00000000 a1=00000000
reg a2=00003000 a3=00004000 a4=00000000 a5=00000000 a6=00000000 a7=00000006
reg usp=00007000 ssp=00000006
```

Note that the contents of address register **a3** were modified by the instruction just executed. Step the program again:

```
M> s 1
```

You will see:

```
00200c@sp  14bc0000    MOVE.B  #000,[A2]
PC = 002010@sp
```

To see the effects of the instruction at location 2010 hex, we will modify register **d0**. Type:

```
M> reg d0=050
```

Now execute the instruction at location 2010. Type:

```
M> s 1
```

You will see:

```
002010@sp  1012          MOVE.B  [A2],D0
PC = 002012@sp
```

This should have cleared **d0**, since the instruction at 200c hex cleared the location pointed to by **a2**. In addition, the zero flag in the status register should be set as a result of moving zeroes into **d0**. Verify this by typing:

```
M> reg d0 st (note you can put more than one
register name on a line)
```

You will see:

```
reg d0=00000000
reg st=2004
```

Next, execute the compare instruction at location 2012 by typing:

```
M> s 1
```

You will see:

```
002012@sp  0c000000    CMPI.B  #000,D0
PC = 002016@sp
```

Look at the register contents:

```
M> reg
```

You will see:

```
reg pc=00002016 st=2004 d0=00000000 d1=00000000 d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00000000 a1=00000000
reg a2=00003000 a3=00004000 a4=00000000 a5=00000000 a6=00000000 a7=00000006
reg usp=00007000 ssp=00000006
```

Notice that the status register zero flag is still set after the compare. As a result, the next instruction, a "branch if equal to zero", should return the program counter to the beginning of the READ_INPUT loop. Type:

```
M> s 1
```

You will see:

```
002016@sp  67f8          BEQ.B   0002010
PC = 002010@sp
```

Note that the program counter did return to the location of READ_INPUT.

You can use the register modification ability to affect the processing of such a loop. This can be handy if you're checking program logic and don't have external hardware to change the value of an input port, or don't want to wait for a loop to time out. Since we're back at READ_INPUT, let's try it. Type:

```
M> reg d0=41
```

Verify the register modification by typing:

```
M> reg d0
```

You will see:

```
reg d0=00000041
```

Now step through the loop:

```
M> s 1
```

You will see:

```
002012@sp  0c000000     CMPI.B  #000,D0
PC = 002016@sp
```

The compare instruction should have reset the zero flag in the status register since the value of **d0** was not zero. Type:

```
M> reg st
```

You will see:

```
reg st=2000
```

As expected, the zero flag is reset. Now step the branch instruction:

```
M> s 1
```

You will see:

```
002016@sp  67f8          BEQ.B   0002010
PC = 002018@sp
```

Notice that the program has now "fallen through" the READ_INPUT loop and is ready to execute the next instruction at PROCESS_COMM.

The only <REG_CLASS> parameter supported by the 68000 emulator is * (all). Therefore, typing **reg** and **reg *** produce the same results.

```
M> reg *
```

You will see:

```
reg pc=00002010 st=2004 d0=00000000 d1=00000000 d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00000000 a1=00000000
reg a2=00003000 a3=00004000 a4=00000000 a5=00000000 a6=00000000 a7=00000006
reg usp=00007000 ssp=00000006
```

Other HP 64700 Emulators provide multiple register classes. For example, on the HP 64700 emulator for the Z80 you could type:

```
M> reg *
```

This would display all the main registers:

```
reg a=ff f=a8 bc=0bff de=0000 hl=0001 ix=0000 iy=0000 sp=003c pc=7f13
```

Or, you could display the alternate register set by typing:

```
M> reg alt
```

You will see:

```
reg a'=00 f'=00 bc'=0000 de'=0000 hl'=0000
```

To display the Z80 interrupt registers, type:

```
M> reg int
```

You will see:

```
reg i=00 iff2=00 imode=00
```

By typing

```
M> reg all
```

You can display the complete register set:

```
reg a=ff f=a8 bc=0bff de=0000 hl=0001 ix=0000 iy=0000 sp=003c pc=7f13 r=46
reg a'=00 f'=00 bc'=0000 de'=0000 hl'=0000 i=00 iff2=00 imode=00
```

Related Commands

s (step; allows you to step through program execution -- combination with the **reg** command is useful in debugging)

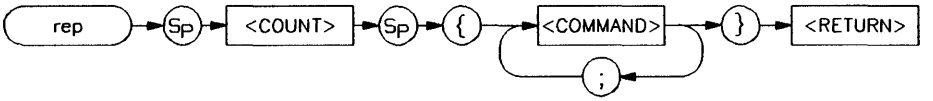
Notes

rep

Summary

Repeat a group of HP 64700 commands

Syntax



Function

The **rep** command allows you to repeat a group of commands a specified number of times. The command list is simply a group of valid HP 64700 commands separated by semicolons and delimited by braces.

Note

Command macros that you define using the **mac** command can be used within a command group for repetition.

No other command input will be accepted until the command group has executed the indicated number of repetitions.

Parameters

<COUNT> An integer value specifying how many times the command list should be executed. A count of zero is a special case, meaning "repeat forever" (the repetition can be terminated by entering <CTRL>-C, which issues a break signal to the emulator).

<COMMAND> Any valid HP 64700 Emulator command, including previously defined macros, may be specified with the options appropriate to the command. The list of commands must be

preceded by an opening brace and followed by a closing brace. Also, the commands must be separated by semicolons. The commands will be executed in the same order as they are specified on the command line.

Defaults

None -- both a count and at least one command must be specified.

Examples

The following example will show you how to simulate a repetitive display of a memory block using ANSI terminal escape sequences, HP 64700 macros, and the **rep** command.

You need to load the program from Appendix A into the emulator. A "priming" run is necessary to make sure all of the pointer registers are set up correctly. Type:

```
M> r 2000
```

Now break the emulator into the monitor. Type:

```
U> b
```

You will need to modify the JMP CLEAR instruction to a JMP PROCESS_COMM instruction so the emulator will repeatedly execute the output routine. Type:

```
M> m 2071=10
M> m 3000=41
```

Next, you will set up two macros. The first will home the cursor and clear the screen on an ANSI standard terminal; the second homes the cursor, displays a memory block, then waits for 0 seconds (the actual wait time is that required to process the command).

```
M> mac cls={echo \1b \5b \32 \4a}
M> mac mem={echo \1b \5b \31 \3b \31 \48; m
-db 4000..401f; w 0}
```

Now start an emulation run at the PROCESS_COMM routine (if you start it earlier the input location of 3000 hex will be cleared; the output memory locations will not change as desired).

```
M> r 2010
```

Clear the terminal screen by typing:

```
U> cls
```

Now, you can start a continuous repetitive display of the output memory block by typing:

```
U> rep 0 mem
```

You will see the following display at the top of your screen. The memory locations from 4000 through 4011 hex will change continuously as they are alternately cleared and written by the sample program.

```
004000..00400f      00 48 00 00 20 49 00 00 4d 00 00 53 00 00 45 00
004010..00401f      00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
waiting for 0 seconds...
echo \1b \5b \31 \3b \31 \48 ; m -db 4000..401f ; w 0
```

To stop the command, enter **<CTRL> C**.

Related Commands

mac (allows assignment of a name to a command group for easy recall of a specified command sequence)

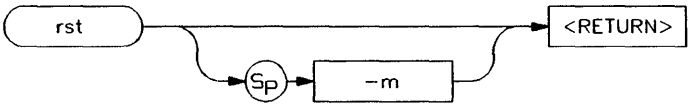
Notes

rst

Summary

Reset the emulation microprocessor

Syntax



Function

The **rst** command resets the emulation microprocessor. An option allows you to specify that the processor should begin executing the emulation monitor code immediately after the reset. If **-m** is not specified, the emulation processor remains in the reset state. Note that any commands which require the emulation processor to execute the monitor code for command processing will not execute while the processor is in the reset state; these include commands such as **reg**.

Commands or hardware signals which will take the emulator out of a reset state include **b**, **r**, **s**, and the CMB /EXECUTE pulse.

Parameters

-m	Causes the emulator to begin executing monitor code immediately after the reset.

Defaults

Reset and remain in the reset state.

Examples

To reset the processor and keep it in the reset state, type:

```
M> rst
```

You will see:

```
R>
```

To reset the processor and have it immediately commence emulation monitor execution, type:

```
U> rst -m
```

You will see:

```
M>
```

Related Commands

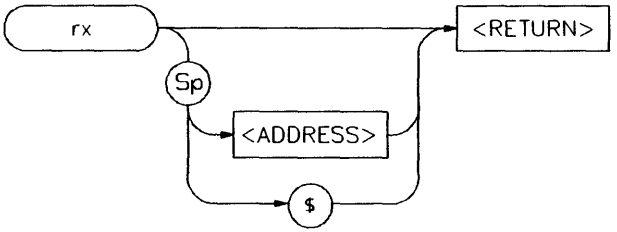
None

rx

Summary

Specify starting address for emulation run upon CMB execution

Syntax



Function

The **rx** command allows you to set the starting address for synchronous CMB (Coordinated Measurement Bus) execution.

If the HP 64700 emulator is connected to the CMB, and the CMB-EXECUTE pulse is detected, followed by the CMB-READY line in the true state, the emulator will begin execution at the address specified by the **rx** command. If no **rx** command has been issued, execution begins at the current program counter value (same as **rx $**).

Execution will begin at the address specified by **rx** every time the conditions listed above are met. For example, if you type the command **rx 100**, the emulator will start executing at address 100 hex every time the CMB-EXECUTE line is pulsed.

The **rx** command automatically turns on CMB interaction by effectively performing the equivalent of a **cmb -e** command whether or not you have done so.

Parameters

<ADDRESS> The <ADDRESS> parameter specifies where to start program execution when the CMB EXECUTE pulse is detected. If $ is specified for address, the current program counter value is used (default). The <ADDRESS> parameter numeric base default is hexadecimal; other bases can be specified with the proper extension. (See the **expr** syntax pages for a description of supported bases.) Some microprocessors, such as the MC68000, support address extensions (function codes or other types of extensions). These extensions may be used in specifying <ADDRESS>.

For specific information on how to specify <ADDRESS> for a given microprocessor, refer to the Emulator User's Guide for your emulator.

Defaults

If you enter the **rx** command without any address parameters, the current address value setting is displayed. If no rx command has been entered since initialization of the emulator, then the default setting is **rx $**.

Examples

To view the current address setting specified by **rx**, type:

```
M> rx
```

You will see:

```
rx $
```

If you want the emulator to begin executing a certain piece of code that begins at address 2000 hex when the CMB-EXECUTE pulse is received, type:

```
M> rx 2000
```

To view the new value of **rx**, type:

```
M> rx
```

You will see:

```
rx 002000
```

Now return the **rx** setting to the default setting by typing:

```
M> rx $
```

Verify the changes:

```
M> rx
```

You will see:

```
rx $
```

To see what value would be used as $ if the CMB-EXECUTE pulse was received, type:

```
M> reg pc
```

You will see something similar to the following (the address will most likely be different on your system):

```
reg pc=0000044e
```

Note that this address would only be used if the processor was reset or running in the emulation monitor. If a user program run occurs before the CMB-EXECUTE line is pulsed, the program counter value is likely to be different.

Related Commands

cmb (enables or disables CMB interaction)

x (initiates a synchronous CMB interaction by pulsing the CMB-EXECUTE line)

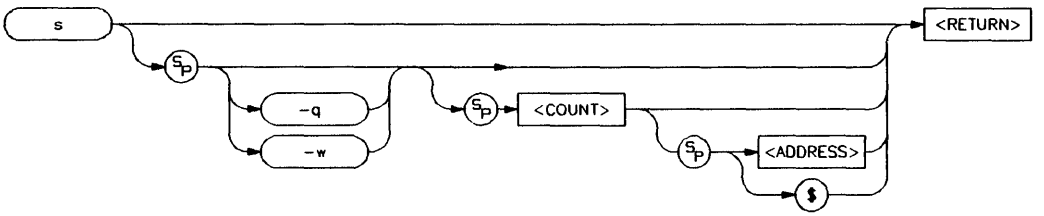
Notes

s

Summary

Step the emulation processor one or more instructions

Syntax



Function

The **s** command allows you to single-step the emulation processor through a program. You can specify the number of steps to execute at a single time; or, you can direct the emulator to step continuously. In addition, you may specify the starting address for stepping.

If the emulator was in the run state (U> prompt) executing a user program when you request the step, it will break to the monitor program before executing the step.

Note

When the Coordinated Measurement Bus (CMB) is being actively controlled by another emulator, the step command (**s**) does not work correctly. The emulator may end up running in user code (NOT stepping). Disable CMB interaction (**cmb -d**) while stepping the processor.

Parameters

-q — If you enter the **-q** parameter, stepping will occur in quiet mode; that is, the instructions

and program counter are not displayed upon execution of each step.

-w

If you enter the **-w** parameter, stepping will be done in whisper mode; only the final program counter value is displayed after the step is executed.

<COUNT>

The <COUNT> parameter allows you to specify the number of steps to execute in sequence before returning command control. For example, if you specify **s 5**, then five instructions will be executed in sequence.

The default base for <COUNT> is decimal. Other number bases may be specified; see the **EXPR** syntax pages for more information.

If you do not specify a value for <COUNT>, then a value of one (1) is assumed. If you specify a step count of zero (0), the emulator interprets this as "step continuously". Continuous stepping can be aborted with the <CTRL>-C command; or, it will be terminated upon receipt of an emulation break condition such as a write-to ROM.

<ADDRESS>

The <ADDRESS> parameter allows you to specify the starting address for stepping. The default is a hexadecimal value; see the **EXPR** syntax pages for information on specifying other number bases. Some microprocessors support address extensions such as function codes; these can generally be specified as part of the <ADDRESS> parameter. Refer to the *Emulator User's Guide* for your emulator for details.

If you substitute $ for the <ADDRESS> parameter, the current program counter

value will be used as the <ADDRESS> value. The same will occur if no address parameter is specified.

Note

If you specify a value for <ADDRESS>, then you must specify a value for <COUNT>. Otherwise, the address value will be interpreted as a step count; the emulator will step the number of locations specified.

Defaults

If you specify s with no parameters, the processor is stepped one instruction from the current program counter location. If you specify <COUNT> but not <ADDRESS>, then the current program counter value is specified for <ADDRESS>.

Examples

The following examples use the 68000 sample program from Appendix A of this manual.

If you want to step 1 instruction from the program's start at address 2000 hex, type:

```
M> s 1 2000
```

You will see:

```
0002000  2479000010  MOVEA.L 0001000,A2
PC = 0002006@sp
```

Or, you could step all the way up to the READ_INPUT routine by typing:

```
M> s 3 2000
```

You will see:

```
0002000  2479000010  MOVEA.L 0001000,A2
0002006@sp  2679000010  MOVEA.L 0001004,A3
000200c@sp  14bc0000    MOVE.B  #000,[A2]
PC = 0002010@sp
```

Note that in both instances, the address and both the hexadecimal and mnemonic version of the instruction are displayed, along with the program counter value after the step. If you want to view all of the processors register's after the step, type:

```
M> reg
```

You will see:

```
reg pc=00002010 st=2004 d0=00000000 d1=00000000 d2=00000000 d3=00000000
reg d4=00000000 d5=00000000 d6=00000000 d7=00000000 a0=00000000 a1=00000000
reg a2=00003000 a3=00004000 a4=00000000 a5=00000000 a6=00000000 a7=00000006
reg usp=00007000 ssp=00000006
```

This allows you to verify that the information from the data area at 1000 hex and 1004 hex was moved to the A2 and A3 registers.

You can also step through the program in "quiet" mode. This inhibits the display of any information about the stepping process. Type:

```
M> s -q 3 2000
```

This may be useful if you wish to step several locations but aren't interested in what's occurring during the step process.

Or, you can step the processor and display only the final program counter value after the step by using the "whisper" mode. Type:

```
M> s -w 3 2000
```

You will see:

```
PC = 0002010@sp
```

You could use this option if you wanted to step several locations and verify only the final program counter value. (For example, you might want to know if the emulator reached a specific location within a certain number of instructions.)

Remember that you must specify a step count value if you specify an address. If you don't, a <CTRL>-C will abort the stepping. Type:

```
M> s 2000
```

You will see: (note that the stepping starts at address 2010, not 2000)

```
0002010@sp  1012        MOVE.B  [A2],D0
0002012@sp  0c000000    CMPI.B  #000,D0
0002016@sp  67f8        BEQ.B   0002010
0002010@sp  1012        MOVE.B  [A2],D0
0002012@sp  0c000000    CMPI.B  #000,D0
0002016@sp  67f8        BEQ.B   0002010
0002010@sp  1012        MOVE.B  [A2],D0
0002012@sp  0c000000    CMPI.B  #000,D0
0002016@sp  67f8        BEQ.B   0002010
0002010@sp  1012        MOVE.B  [A2],D0
PC = 0002012@sp
```

(enter **<CTRL>-C**)

```
!STATUS  686! Stepping aborted; number steps completed: 10
```

You can also use the step capability in stepping through a loop. Type:

```
M> s 3 2010
```

You will see:

```
0002010@sp  1012        MOVE.B  [A2],D0
0002012@sp  0c000000    CMPI.B  #000,D0
0002016@sp  67f8        BEQ.B   0002010
PC = 0002010@sp
```

You can then modify some value which affects the loopback logic and watch its effects on the processor instruction execution. Type:

```
M> m 3000=41
```

(This inputs a "command A" to the sample program.)

Now type:

```
M> s 4
```

```
0002010@sp  1012        MOVE.B  [A2],D0
0002012@sp  0c000000    CMPI.B  #000,D0
0002016@sp  67f8        BEQ.B   0002010
0002018@sp  0c000041    CMPI.B  #041,D0
PC = 000201c@sp
```

Note that the program exited the loop and has begun processing a command.

You can assign values to label names using the **equ** command and then use these labels in specifying step information. For example, the number of instructions in the READ_INPUT loop is 3, it begins at 2010 hex. Type:

```
M> equ readcount=3
M> equ readinput=2010
```

Now type:

```
M> m 3000=00
```

(This modifies the command input area of the program to a null value.)

Now you can step through the loop, one iteration at a time:

```
M> s readcount readinput
```

You will see:

```
0002010  1012          MOVE.B  [A2],D0
0002012@sp  0c000000     CMPI.B  #000,D0
0002016@sp  67f8         BEQ.B   0002010
PC = 0002010@sp
```

If your emulator supports symbols, and you have symbols loaded, when you execute a **step** command, you will see the module and symbol in the output. For example, using the 8051 emulator, you may execute the following command:

```
R> s 1 CMD_RDR:INIT
```

The result will resemble:

```
03000@x  CMD_RDR:INIT     MOV SP,#fe
PC = 03003@p
```

Related Commands

r (run emulation processor from a specified address)

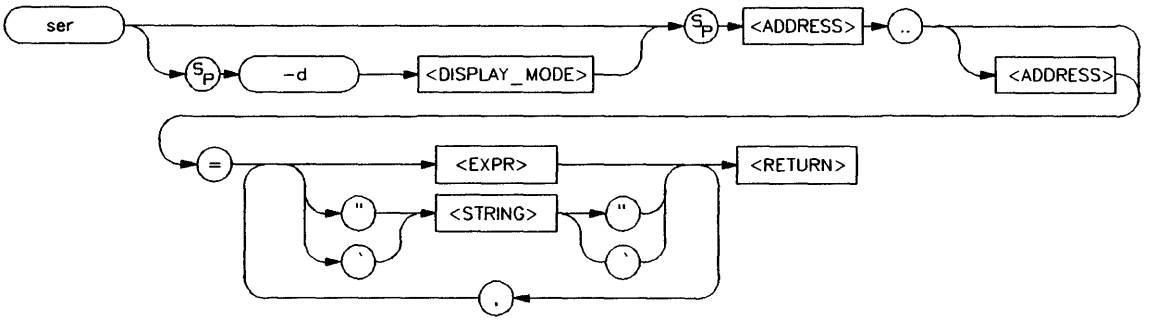
reg (view or modify processor register contents)

ser

Summary

Search emulation or target memory for values

Syntax



Function

The **ser** command allows you to search memory for a data value, a character string, or a combination of both. For every pattern match, the starting address of the match is displayed.

Using the **-d** (display mode) option, the method of interpreting the pattern supplied by the user can be altered. If no option is given, the display mode used is taken from global default set by the **mo** command.

If addresses specified in the search reside in target system memory, the emulator is broken to the monitor and returned to the user program when the command is completed.

Parameters

-d

The **-d** operator, in combination with the <DISPLAY_MODE> parameter, allows you to specify the display mode used for the search. As a result, you can alter the method

	used by the system for interpreting the display list data and the resultant matches.
<DISPLAY_ MODE>	This is a single character specifying the display mode to be used in the search. Most processors support **b**, for byte; some processors optionally support **w** (word) and **l** (long word). For specific information on the <DISPLAY_MODE> parameters supported by your emulator, refer to the *Emulator User's Guide.*
<ADDRESS>	You use <ADDRESS> to specify first the lower, and possibly the upper, address boundaries of the memory range to search for the given data pattern. <ADDRESS> defaults to a hexadecimal number; expressions may also be provided. In addition, certain emulators support additional processor specific addressing information such as function codes. Refer to the *Emulator User's Guide* for your emulator for further details.
..	The two periods (..) are used as a separator between the lower and upper address boundary specifications. Notice that no additional spaces are inserted. You can use "<ADDRESS>.." to specify the range from the address through the next 127 bytes.
<EXPR>	<EXPR> is a numeric expression to be used as a reference pattern in the search. The default is a hexadecimal number; other bases and expressions may be specified. Refer to the <EXPR> syntax for further information.

<STRING> You specify <STRING> if you want to search for an ASCII character pattern. Note that <STRING> must be bounded by single open quote marks (') or double quotes (").

Note 

Many keyboards (and printers) actually represent the single open quote mark ' as an accent grave mark. In any case, the correct key is the one which produces a character encoded as ASCII 60 hexadecimal. The correct double quote mark is the character encoded as ASCII 22 hexadecimal.

Note 

If the character string you are searching for contains double quotes, you must delimit the string with single open quotes and vice versa. For example, the string **"Type "C""** will return an error; the string **'Type "C"'** is correct.

Note 

You can concatenate various combinations of <STRING> and <VALUE> to form more complex search patterns by separating the parameters with commas (,).

Defaults

At least one address range and data pattern must be specified. If no display mode is set with the **-d** option, the current global display mode from the **mo** command is used.

Examples

We will do some searches of the message area from the sample program in Appendix A. Let's look at the message area in byte form first. Type:

```
M> m -db 1000..103f
```

You will see:

```
001000..00100f      00 00 30 00 00 00 40 00 54 48 49 53 20 49 53 20
001010..00101f      4d 45 53 53 41 47 45 20 41 54 48 49 53 20 49 53
001020..00102f      20 4d 45 53 53 41 47 45 20 42 49 4e 56 41 4c 49
001030..00103f      44 20 43 4f 4d 4d 41 4e 44 ff ff e7 f7 fd fe ff
```

Now let's search for the ASCII character string "THIS" (which consists of the values 54,48,49, and 53 hexadecimal). Type:

```
M> ser 1000..103f="THIS"
```

You will see:

```
pattern match at address: 001008
pattern match at address: 001019
```

Note the correspondence of the first character found in the sequence with the addresses of the data area shown above.

You can also combine searches for numeric values, numeric expressions, and ASCII strings. Type:

```
M> ser -db 1000..103f=20,"MESSAGE",10+10
```

You will see:

```
pattern match at address: 00100f
pattern match at address: 001020
```

Now we will modify the display mode and notice the effect it has upon the search. First, let's look at the memory block in the new mode. Type:

```
M> m -dw 1000..103f
```

You will see:

```
001000..00100f      0000 3000 0000 4000 5448 4953 2049 5320
001010..00101f      4d45 5353 4147 4520 4154 4849 5320 4953
001020..00102f      204d 4553 5341 4745 2042 494e 5641 4c49
001030..00103f      4420 434f 4d4d 414e 44ff ffe7 f7fd feff
```

Now search for the same string, but change the display mode, by typing:

```
M> ser -dw 1000..103f=20,"MESSAGE",20
```

The search failed since the end of the expression was not on a word boundary. (It would have been satisfied if the first and last 20's resided on word boundaries and if we searched for an additional space at the end of the word "MESSAGE " that is, the word pattern 0020 4d45 5353 4147 4520 0020.) Now let's do another search to illustrate the effects of the display mode. Type:

```
M> ser -dw 1000..101f=0003
```

Again, the search is unsatisfied because 03 hex is not at the end of a word boundary. Now type:

```
M> ser -dw 1000..101f=0030
```

You will see:

```
pattern match at address: 001001
```

Now the pattern is found since it is on a word boundary at address 1001.

Now look at the same search patterns in long word display mode. View the block by typing:

```
M> m -dl 1000..103f
```

You will see:

```
001000..00100f     00003000 00004000 54484953 20495320
001010..00101f     4d455353 41474520 41544849 53204953
001020..00102f     204d4553 53414745 2042494e 56414c49
001030..00103f     4420434f 4d4d414e 44ffffe7 f7fdfeff
```

Search for the pattern 030 hex by typing:

```
M> ser -dl 1000..101f=030
```

It isn't found since it isn't on a word boundary. Now type:

```
M> ser -dl 1000..101f=03000
```

You will see:

```
pattern match at address: 001000
```

The match is found at the first location in the range.

Related Commands

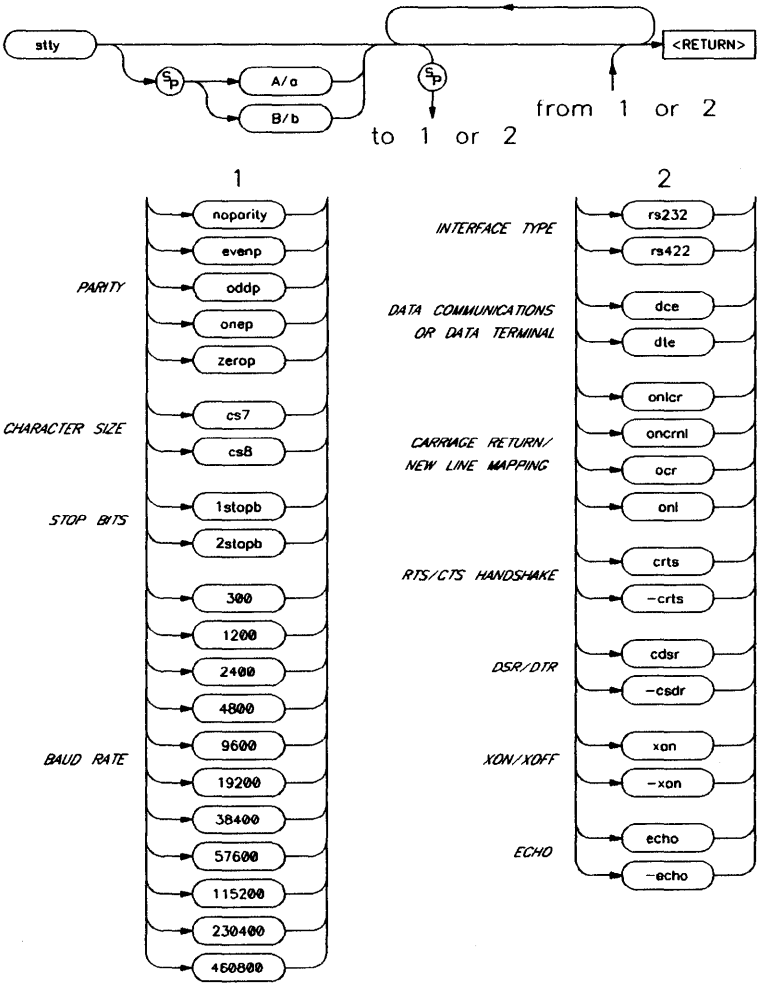
cp (used to copy the contents of one memory range to another)

m (used to display/modify memory locations)

stty

Summary Set data communications parameters

Syntax



Function

The **stty** command allows you to modify the parameters of the data communications ports without changing the configuration switch settings.

For example, you may have a situation where a terminal is controlling the emulator through port B and a modem is attached to port A for communications with a host computer. The **stty** command allows you to vary the baud rate of port B easily so that you can use a high speed while operating the emulator and a low speed while in transparent mode, using the modem to communicate with the host. The advantage here is that you don't have to change the configuration switches and issue an **init** command to read them; your current emulator and analyzer configurations are preserved.

Either port A or B may be modified by **stty**.

Note

For further information on the meanings of various data communications parameters, you may refer to the book entitled *Touring Datacomm: A Data Communications Primer.* This book is orderable from HP's Direct Marketing Division under the part number 5957-4622. Another book which may be helpful is *The RS-232 Solution*, orderable from HP under the product number 92234X. You also may need to refer to the hardware and software reference manuals that are supplied with your terminal and/or host computer for further information on required data communications parameters for links to those devices.

Parameters

PARITY	Parity for either port may be set odd, even, zero, one, or none.
CHARACTER SIZE	The length of each character sent by the system may be set to 7 bits or 8 bits.

STOP BITS	The number of stop bits used to terminate each character may be set to one (1) or two (2).
BAUD RATE	The baud rate (rate at which bits are transmitted and received) may be set to one of the following values: 300 1200 2400 4800 9600 19200 38400 57600 115200 230400 460800.

Note 

The maximum baud rate setting for Port B is 38400 baud.

INTERFACE TYPE	The type of interface on port A may be set to either RS-232 or RS-422.

Note 

Port B is fixed as an RS-232 device.

RS-422 utilizes balanced transmission lines and therefore can achieve much higher data rates with reliability over long distances than RS-232. Otherwise, the interfaces are similar.

DATA COM-MUNICATIONS OR DATA TERMINAL	Port A may be set to operate either as Data Communications Equipment (DCE) or as Data Terminal Equipment (DTE). This configures the handshake lines and transmit/receive lines for the proper signal to pin relationships on the interface.

Note

Port B is permanently configured as Data Communications Equipment (DCE). In many cases it will require a null modem connector to achieve proper communications with a host computer. Refer to the *HP 64700 Hardware Installation and Configuration* manual for further information.

CARRIAGE RETURN/ LINE FEED MAPPING	You can select several different options for terminating lines of output from the system, depending on what is required by your hardware. The following choices are available: **onlcr** -- generate new-line and carriage-return on output **ocrnl** -- generate carriage-return and new-line on output **ocr** -- generate carriage-return on output **onl** -- generate new-line on output
RTS/CTS HANDSHAKE	The option **crts** enables the Request To Send/Clear To Send handshake. Specifying **-crts** disables this handshake.
DSR/DTR STATUS	The option **cdsr** enables exchange and recognition of the Data Set Ready/Data Terminal Ready status lines. Specifying **-cdsr** disables the exchange.

XON/XOFF HANDSHAKE

If you specify **xon**, the system generates XON/XOFF (DC1/DC3 characters) software handshaking to control the amount of data received at a given time. Specifying **-xon** disables this handshake sequence.

(When the emulator's receive buffer is full, it will send a DC3 (XOFF) character to the host to stop transmission; when it is ready for more data, it will send a DC1 (XON) character to restart transmission.)

Note

If you toggle the **xon** parameter when running at 1200 baud and below, the **stty** command will return invalid characters. The PC Interface attempts to do this when starting up and fails with a datacomm error. To get around this problem, set switch 13 on the emulator's back panel (enable xon) to allow the PC Interface to start up successfully. In the Terminal Interface, just enter another carriage return to regain proper communications.

ECHO

If you specify **echo**, all characters received by the emulator datacomm are echoed back to the sending system. Specifying **-echo** means the system will not echo back characters received.

You will normally use this in conjunction with the echo settings required by your host computer and your terminal. Most Hewlett-Packard systems will require that you enable the echo feature, as HP host computers automatically echo characters back to data terminal devices.

Defaults

If no ports are specified, the current settings of both ports are displayed; if a port designator is supplied with no other parameters, the current settings for that port are displayed. The powerup default configurations for each port are determined by the rear panel configuration switches; refer to the *HP 64700 Emulator Hardware Installation and Configuration* manual for more information.

Examples

To display the current data communications setting for both ports, type:

```
M> stty
```

You will see:

```
stty A  9600  cs8  1stopb  noparity  dce  rs232  -crts  -cdsr  xon  onlcr  echo
stty B  9600  cs8  1stopb  noparity  rs232  -crts  -cdsr  xon  onlcr  echo
```

Now, set the baud rate for port B to 1200 baud by typing:

```
M> stty B 1200
```

To view the changed baud rate, type:

```
M> stty B
```

You will see:

```
stty B  1200  cs8  1stopb  noparity  rs232  -crts  -cdsr  xon  onlcr  echo
```

Let's change the baud rate back to 9600 and disable local echo on Port B at the same time. Type:

```
M> stty B 9600 -echo
```

Verify the changes by typing:

```
M> stty b
```

Note that the port designator can also be supplied in lower case. You will see:

```
stty B  9600  cs8  1stopb  noparity  rs232  -crts  -cdsr  xon  onlcr  -echo
```

Now let's delete the XON/XOFF software handshake and add the RTS/CTS hardware handshake. Type:

```
M> stty B -crts -xon
```

Again, verify the changes by typing:

```
M> stty B
```

```
stty B  9600  cs8  1stopb  noparity  rs232  crts  -cdsr  -xon  onlcr  -echo
```

Related Commands

po (allows you to specify whether a given port is to be used for standard input, standard output, standard error, or some combination thereof)

xp (used to enable/disable transparent mode communications and set the command escape character for transparent mode)

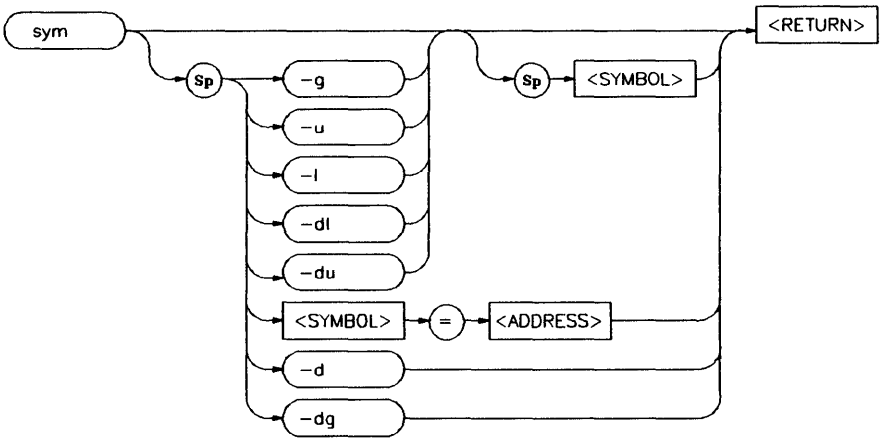
Notes

sym

Summary

Manage the emulator symbol table

Syntax



Function

The **sym** command defines, displays, or deletes symbols in the emulator.

Three types of symbols are supported: global, local, and user. Global symbols reference addresses anywhere in memory using an absolute reference. Local symbols also use absolute addressing but are grouped within a "module." User symbols are defined at the command line. Global and local symbols cannot be defined at the command line.

The definition of a module for grouping local symbols depends on the environment being used. For local symbols created by a high-level language, a module might be a function, a procedure, or a separately compilable source file. When you define local symbols through the use of a symbol file, a module, in effect, becomes a technique to manage the symbols. It can be a mnemonic device to refer to modules, or it can be a simple way to group local symbols into a set for display and deletion purposes since the **sym** command facilitates manipulation of local symbols by their module name.

Symbols are used like equated variables. When using symbols in expressions, only the + and - operators can be used immediately before and after the symbol name. The expression can contain literals and equated (**equ**) labels, but not other symbols.

When using symbols, if a symbol and an equated value have the same name, the equated value will be used.

The symbol table can be updated in three ways:

- You can enter user symbols at the command line.
- You can update it from an external "symbol file" using the **load -So** command.
- You can load an absolute file (such as an Intel OMF file) which can contain symbols as well as program code.

A "symbol file" is a text file containing user-specified symbols. Refer to a discussion of the symbol file in appendix A.

Note 

The **sym** command presently applies to some HP 64700-Series emulators, and may apply to all HP 64700-Series emulators in the future. If you are using an emulator that does not presently support symbols, when you try to execute the **sym** command, a message will be displayed indicating that symbols are not supported on your emulator.

If your emulator firmware is less than version A.02.00, you will not be able to use the **sym** command because your emulator will not support symbols. To verify the version number of your emulator firmware, execute "ver" at the Terminal Interface prompt.

Even if your emulator firmware version is A.02.00 or greater, your HP 64700-Series emulator may not necessarily support symbols.

Parameters

<ADDRESS>	The <ADDRESS> parameter specifies the value to assign to a user symbol.
-d	The **-d** option deletes all symbols.
-du	The **-du** option deletes user symbols. If a **<NAME>** parameter is not included, all user symbols are deleted. If a **<NAME>** parameter is included, only user symbols matching the entered name are deleted.
-dg	The **-dg** option deletes all global symbols. No option exists to delete one global symbol.
-dl	The **-dl** option deletes local symbols in a module. If a **<NAME>** parameter is not included, all local symbols are deleted for all modules. If a **<NAME>** parameter is included to specify a module name, only local symbols in the module matching the entered name are deleted.
-g	The **-g** option specifies the display of global symbols. If a **<NAME>** parameter is not included, all global symbols are displayed. If a **<NAME>** parameter is included, only global symbols matching the entered name are displayed.
<NAME>	This represents the symbol label to be defined or referenced. The format of the symbol name reference is determined by the type of symbol, where: **name** is a user symbol or module name **:name** is a global symbol name **name:** is a local module name

module:name is a symbol name in a local module.

In addition, symbols can be referenced using a "wild card" expression when displaying and deleting names. Only one wildcard character can appear in a symbol name. An asterisk ("*") character is used to represent zero or more characters at the end of a symbol name. A wildcard can be used in any of the following symbol types:

name* represents a user symbol name followed by zero or more of any character or characters

:name* represents a global symbol name followed by zero or more of any character or characters

module:name* represents a local module:symbol followed by zero or more of any character or characters.

-l This option allows you to display local modules and symbols. If a **<NAME>** parameter is not included, all local modules are displayed. If a **<NAME>** parameter is included, only local symbols matching the symbol name or module are displayed.

-u This option allows you to display user symbols. If a **<NAME>** parameter is not included, all user symbols are displayed. If a **<NAME>** parameter is included, only user symbols matching the entered name are displayed.

Defaults

The **sym** command without any parameters displays all of the symbols currently defined.

Examples

To display all symbols, enter:

```
M> sym
```

To display all global symbols, enter:

```
M> sym -g
```

To display a global symbol, enter:

```
M> sym -g :GLOB_SYM
```

To display a user symbol, enter:

```
M> sym -u mysymbol
```

To display all local modules, enter:

```
M> sym -l
```

To display symbols in a local module, enter:

```
M> sym -l LOCAL_MOD:
```

To display a symbol in a local module, enter:

```
M> sym -l MOD_NAME:SYM_NAME
```

To delete all global symbols, enter:

```
M> sym -dg
```

To define a user symbol, enter:

```
M> sym mysymbol=107h
```

To display all symbols or local modules whose names begin with "symb", enter:

```
M> sym symb*
```

Related Commands

equ (used to equate names to expressions)

load (used to load a program file with symbols, or a symbol text file)

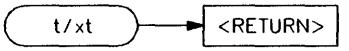
Notes

t,xt

Summary

Start an analyzer trace

Syntax



Function

The **t** and **xt** commands start emulation and external traces, respectively. These commands (or **tx** if making a synchronous CMB execution) must be entered to actually begin a measurement; most other trace commands are used only for specification of triggering, sequencer, and storage parameters; or to display trace results or status.

If the external analyzer has been linked to the emulation analyzer via the **xtmo** command, the **xt** command is invalid and both analyzers begin a trace when the **t** command is entered.

Parameters

None.

Defaults

Does not apply.

Examples

To begin a trace, enter:

```
M> t
```

You will see:

```
Emulation trace started
```

To halt a trace in process, enter:

```
M> th
```

You will see:

```
Emulation trace halted
```

Related Commands

r (starts a user program run; normally will be specified after entering the **t** command)

th (halts a trace in process)

ts (allows you to determine the current status of the emulation analyzer)

tx (specifies whether a trace is to begin upon start of CMB execution)

x (begins synchronous CMB execution)

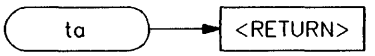
xtmo (specifies whether or not the external analyzer bits are to be treated as a separate analyzer or integrated with the emulation analyzer. If associated with the emulation analyzer, the **xt** command is invalid; the **t** command starts the trace on both analyzers.)

ta

Summary

Display analyzer line activity

Syntax



Function

The **ta** command allows you to display the activity on each of the analyzer input lines. Each signal may be low, high, or moving. These are displayed as follows:

Type of Signal Activity	Symbol Displayed
Signal is Low	0
Signal is High	1
Signal is Moving	?

Each pod (group of 16 lines) is displayed on a single line with bit 0 (LSB) at the far right and bit 15 (MSB) on the far left. Each pod represents the following analyzer bits:

Pod	Emulation Analyzer Bits	External Analyzer Bits
Pod 1	Emulation Bits 0 thru 15	None
Pod 2	Emulation Bits 16 thru 31	None
Pod 3	Emulation Bits 32 thru 47	None
Pod 4	Emulation Bits 48 thru 63	None

Parameters

None.

Defaults

Does not apply.

Examples

To display the current status of analyzer signal activity, type:

```
M> ta
```

You will see a display similar to the following:

```
Pod 3       = 0??????? ????????
Pod 2       = 11?00110 00000000
Pod 1       = 00000?00 1???????0
```

You can interpret the results as follows:

Bit 15 of Pod 3 is low; all other Pod 3 bits are moving.

Bits 9,10,14 and 15 of Pod 2 are high, bit 13 is moving; all others are low.

Bit 7 of Pod 1 is high; bits 1-6 and 10 are moving; all others are low.

Related Commands

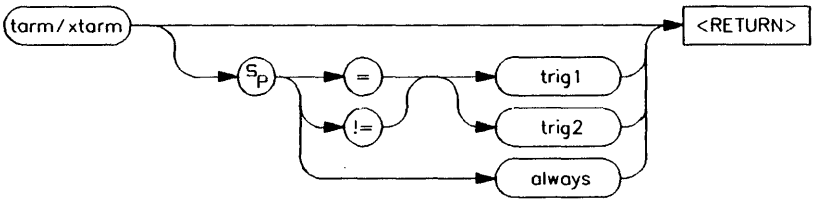
xtv (used to set the threshold voltages for the optional external analyzer inputs; incorrect specification may show up as lack of activity in a **ta** display)

tarm,xtarm

Summary

Specify arming condition for analyzers

Syntax



Function

The **tarm** (**xtarm**) command allows you to specify an arming condition for the emulation and external analyzers. You can specify the arm condition as the assertion of the trig1 or trig2 signals or as **tarm always**. The arm condition may then be used in specifying the analyzer trigger or in specifying branch conditions for the sequencer, as well as count or prestore qualifiers.

If the analyzers are connected through use of the **xtmo** command, then the **xtarm** command is invalid. In this case, the **tarm** command will set the arming condition for the analyzer combination.

Parameters

=, != The operators = and != are used to respectively indicate that the arm condition is equal to or not equal to the specified **trig1** or **trig2** condition.

Note 

If the external analyzer is configured to operate as a timing analyzer (**xtmo -t**) then the != operator is invalid when used in the **xtarm** command as given to the external analyzer. Only the = operator will be recognized.

trig1	If you specify **tarm =trig1** as the arming condition, then the assertion of the trig1 signal will arm the analyzer. Conversely, if you specify **tarm !=trig1**, the analyzer will remain armed until the trig1 signal is asserted. The trig1 signal can be asserted from many sources including the analyzer itself or the rear panel BNC connector or the CMB. See **bnct**, **cmbt**, and **tgout** for examples.
trig2	If you specify **trig2** as the arming condition, then the assertion of the trig2 signal will arm the analyzer. Conversely, if you specify **tarm !=trig2**, the analyzer will remain armed until the trig2 signal is asserted. The trig2 signal can be asserted from many sources including the analyzer itself or the rear panel BNC connector or the CMB. See **bnct**, **cmbt**, and **tgout** for examples.
always	If you specify **tarm always** as the arming condition, then the analyzer is continuously armed.

Defaults

If no parameters are supplied, the current **tarm** condition is displayed. The default setting after powerup or **tinit** is **tarm always**.

Examples

To view the current state of **tarm**, type:

```
M> tarm
```

You will see:

```
tarm always
```

You may want to hook an external instrument, such as a logic analyzer, to the HP 64700 rear panel BNC port and have the external instrument trigger an emulation analyzer trace. Type the following:

```
M> bnct -r trig1
M> tcf -c
M> tarm =trig1
M> tg arm
```

This will cause the emulation analyzer to trigger upon assertion of the rear panel BNC signal. To return the analyzer to the continuously armed state, type:

```
M> tarm always
```

Perhaps you want the analyzer to store only states received while there is NOT a trigger signal on the CMB (Coordinated Measurement Bus). To do this, type:

```
M> cmbt -r trig2
M> tcf -c
M> tarm !=trig2
M> tsto arm
```

Here, we've set the trig2 signal to receive the CMB trigger. Then we set the emulation analyzer configuration to complex (this is required to use the **arm** parameter in analyzer expressions). Next, we set the tarm condition to the logical NOT of the trig2 signal; finally, we qualify analyzer storage with the arm parameter.

Related Commands

bc (can be used to cause the emulator to break to monitor execution upon receipt of the trig1 and/or trig2 signals)

bnct (used to define connections between the internal trig1 and trig2 signals and the rear panel BNC connector)

cmbt (used to define connections between the internal trig1 and trig2 signals and the CMB trigger signal)

tgout (defines whether or not the trig1 or trig2 signals are driven when the analyzer finds the trigger state)

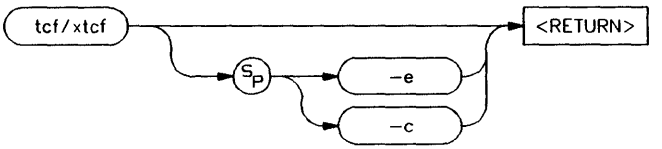
Notes

tcf,xtcf

Summary

Set the analyzer configuration to easy or complex

Syntax



Function

The **tcf** (**xtcf**) commands are used to set the configuration for the emulation (external) analyzer.

There are two possible configurations for the analyzer, an easy configuration (**tcf -e**) and a complex configuration (**tcf -c**). Below, each of the configurations is described briefly, along with some of the commands that modify the analyzer in each configuration. The command descriptions are not meant to be an exhaustive list of each command's features; you should refer to the syntax pages for that particular command.

Easy Configuration

When in easy configuration (**tcf -e**), much of the complexity of the analyzer is hidden from you. Some measurement power is lost; when you need the full power of the analyzer, you can switch to complex configuration.

Expressions. In easy configuration, all analyzer commands take the general form of **<command> <simple_expression>**. The commands that use this form are **tcq, tif, telif, tg, tpq,** and **tsto**. A simple expression is the information that can fit into a single pattern or a single range (see **tpat**, **trng**, and **SIMPLE_EXPR** syntax for further information). Examples are **addr=2105**, **data!=15 or data!=ff**, and **addr=4012..401a**.

Sequencing. The easy configuration allows you to have the analyzer search for a simple expression; when it is found, it can then search for a different simple expression. The ability to search for one expression, then search for another expression based on the first is known as sequencing.

In easy configuration, there are 4 sequencer terms available. Each has a primary sequence branch, which always branches to the next sequencer term (1 to 2, 2 to 3, and so on). The branch out of the last term defines the trigger term. A global restart term is also available, which will return the sequencer to term 1 if found. If both the primary branch and global restart term are satisfied simultaneously, the primary branch is always taken in preference to the restart.

Sequencer Manipulation. The simplest sequencer control is the **tg** command. This defines a one term sequence with the trigger occurring upon the branch out of the term. You can specify an occurrence count; that is, the number of times the given trigger qualifier must be found to satisfy the trigger condition.

You can exercise greater control over the easy configuration sequencer using the **tsq** command. This command allows you to insert additional sequence terms (up to the limit of four) or delete terms.

By using the **tif** command, you can define the primary branch condition for each sequence level. You can also specify an occurrence count for each branch condition. The primary branch out of the last sequence term in the list defines the trigger condition.

The **telif** command specifies the global restart condition. If both a primary branch and global restart condition are satisfied at the same time, the primary branch is always taken. However, if the primary branch has an occurrence count greater than one (1), and the global restart is encountered before the occurrence count is satisfied for the primary branch, the global restart is taken, and the primary branch occurrence count is reset to zero.

Storage Specification. You can specify which events should be stored by the analyzer using the **tsto** command. This is a global storage qualifier; that is, the qualifier is identical for all sequencer terms. Analyzer events that cause the sequencer to change states are always stored, regardless of the storage qualifier.

State/Time Counts. You can set up the analyzer to count time between states or count occurrences of a specific state using the **tcq** command.

Prestore. The analyzer has a two stage prestore pipeline. You set up the qualifier for this pipeline using the **tpq** command. When the qualifier is found, the event is stored in the pipeline; when a real storage event is found (matching the **tsto** qualifier), the pipeline is flushed and placed into trace memory immediately prior to the storage event. You can use the feature to observe the relationships between certain program variables and program routines or between two program routines. (For example, you might set a prestore state to a condition required to execute a specific routine.)

Complex Configuration

The full analyzer capability is available to you in the complex configuration (**tcf -c**). Using the multiple sequence terms, primary and secondary branch capability, and powerful expression capability, you can make just about any conceivable measurement.

Expressions. In complex configuration, all analyzer commands take the general form of **<command> <complex_expression>**. The commands that use this form are **tcq, tif, telif, tg, tpq,** and **tsto**. A complex expression is made up of pattern, range and arm labels, tied together with various operators that define the specific condition. Each of the pattern and range labels must be previously assigned to a specific simple expression using the **tpat** and **trng** commands. (These two commands are only available in the complex configuration.) So, you might define some pattern labels and a range label as follows:

```
U> tpat p1 addr=205a
U> tpat p5 data!=00
U> trng addr=4000..4011
```

And then make complex expressions as follows:

```
p1 or p5
r and p5
p1 | !r
```

See the **<COMPLEX_EXPR>** syntax pages for details on complex expressions.

Sequencing. The complex configuration allows you to have the analyzer search for a complex expression; when it is found, it can then search for a different complex expression.

In complex configuration, there are always 8 sequencer terms. Each has a primary sequence branch, which can branch to any sequencer term (1 to 5, 2 to 8, and so on). A secondary branch is also available, which branch to any sequencer term. If both the primary branch and secondary branch are satisfied simultaneously, the primary branch is always taken in preference to the secondary branch.

Sequencer Manipulation. The simplest sequencer control is the **tg** command. As in easy configuration, this defines a two term sequence with the trigger in the second term. You can specify an occurrence count; that is, the number of times the given trigger qualifier must be found to satisfy the trigger condition.

You can exercise greater control over the complex configuration sequencer using the **tsq** command. Although you cannot add or delete sequence terms in complex configuration (there are always eight), you can specify the trigger term; you can also reset the sequencer (which clears all the branch specifiers and storage qualifiers).

By using the **tif** command, you can define the primary branch condition for each sequence level. You can also specify an occurrence count for each branch condition, and the destination term for each branch.

The **telif** command specifies the secondary branch condition, which can jump to any sequence term. If both a primary and secondary branch condition are satisfied at the same time, the primary branch is always taken. However, if the primary branch has an occurrence count greater than one (1), and the secondary branch is

encountered before the occurrence count is satisfied for the primary branch, the secondary branch is taken, and the primary branch occurrence count is reset to zero.

Storage Specification. You can specify which events should be stored by the analyzer using the **tsto** command. You may specify different storage qualifiers for each sequencer term, allowing you to precisely control the information captured by the analyzer. If you don't specify a term number when specifying the storage qualifier, the storage qualifier for all terms is set to the same qualifier.

State/Time Counts, Prestore. The state/time counting and prestore facilities are identical to those provided in the easy configuration; however, you must specify a complex expression instead of an easy expression in qualifying the state count or prestore.

Resetting the Analyzer Configuration.

When the analyzer configuration is changed, the entire analyzer specification is reset. You can perform a reset back to the default sequencer setup in either configuration by using the **tsq -r** command.

When the trace configuration is changed, the count qualifier (**tcq/xtcq**) is reset to "none" (instead of "time") if the clock mode (**tck/xtck**) is fast (F) or very fast (VF).

Parameters

-e	Specifying **-e** sets the analyzer to the easy configuration.
-c	Specifying **-c** sets the analyzer to the complex configuration.

Defaults

If no parameters are supplied, the current analyzer configuration is displayed. After powerup or **tinit**, the default analyzer configuration is **tcf -e**.

Examples

To display the current analyzer configuration after powerup, type:

```
M> tcf
```

You will see:

```
tcf -e
```

To set the analyzer to complex configuration, type:

```
M> tcf -c
```

Related Commands

tarm (used to set the analyzer arm specification; this specification can only be used in analyzer expressions in complex configuration)

tcq (sets the expression for the trace count qualifier in either analyzer configuration)

telif (sets the global restart in easy configuration, secondary branch condition in complex configuration)

tg (used to set a trigger expression in either analyzer configuration)

tif (sets primary branch specification in either analyzer configuration)

tpat (used to label complex analyzer expressions with a pattern name; the pattern name is then used by the analyzer setup commands. Only valid in complex configuration)

tpq (specifies trace prestore qualifier in either analyzer configuration)

trng (defines a range of values to be used in complex analyzer expressions)

tsto (specifies a qualifier to be used when storing analyzer states)

tsq (used to modify the trace sequencer's number of terms and trigger term)

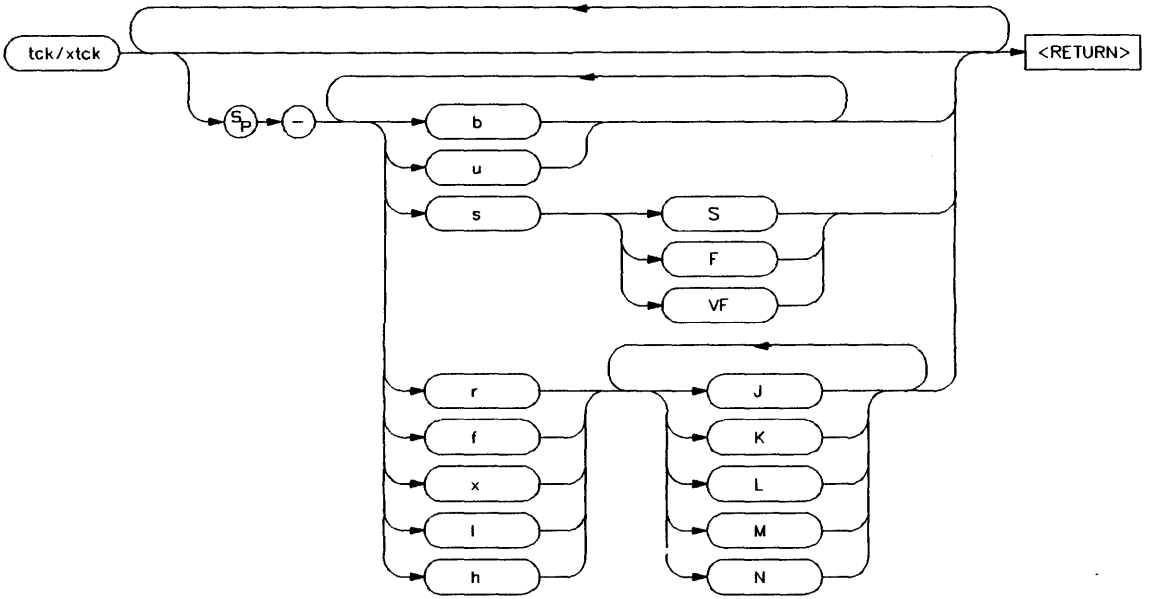
xtmo (used to append or disconnect the external analyzer to/from the emulation analyzer)

tck,xtck

Summary

Specify analyzer master clock qualifiers

Syntax



Function

The **tck** (**xtck**) command allows specification of clock qualifiers, master edges and maximum clock speed of the master clocks used for the emulation and external analyzers.

The **tck** command is included with the system for the purpose of internal system initialization and system control through high-level software interfaces; you should generally not use this command.

If you are using the optional external analyzer, you will use the **xtck** command to set the clock parameters for your external analysis traces.

Changing the clock speed with the **s <SPEED>** option affects the **tcq** command parameters. When speed is set to **sS** (slow), the **tcq** command may either count states or time. When speed is set to **sF** (fast), the **tcq** command may be used to count states but not time. If clock speed is set to **sVF** (very fast), **tcq** cannot count either state or time and should be set to **tcq none.**

The clocking options operate on five different clock signals: **J, K, L, M** and **N**. Clocks **L, M,** and **N** are generated by the emulator; the emulation master clock edges are set at powerup for the particular emulator being used and should not be changed by the user. The **J** and **K** clocks are the clock inputs on the external trace probe (if one is present). These clock signals should only be used to clock the external trace; they should not be used to clock the emulation trace although it may occasionally be useful to use the external clock signals as qualifiers for the emulation trace. The **L** and **M** clocks may also be used to clock the external trace as well as the emulation trace.

When several clock edges are specified, any one of the edges clocks the given trace. If several qualifiers (**l** or **h**) are specified, they are ORed so that the trace is clocked when any of the qualifiers are met.

Parameters

b	If the **b** option is specified, only background monitor code will be qualified into the analyzer.
u	If the **u** option is specified, only user code will be qualified into the analyzer. This is the default.

Note

The **u** and **b** qualifiers are ORed with all of the other qualifiers specified.

s	The **s** option indicates that the maximum clock speed is to be modified per a one or two letter code immediately following.
S	Specifies a clock speed of SLOW; less than or equal to 16 MHz.
F	Specifies a clock speed of FAST; between 16 MHz and 20 MHz.
VF	Specifies a clock speed of VERY FAST; between 20 MHz and 25 MHz.
r	Specifying **r** indicates that the analyzer is to be clocked on the rising edge of the indicated clock signal.
f	Specifying **f** indicates that the analyzer is to be clocked on the falling edge of the indicated clock signal.
x	Specifying **x** indicates that the analyzer should be clocked on both the rising and falling edges of the indicated clock signal.
l	Specifying **l** indicates that the analyzer should only be clocked by other clock signals when this clock signal is low (less positive/more negative voltage). Used as a qualifier (example: clock on rising edge of J only if K is low).
h	Specifying **h** indicates that the analyzer should only be clocked by other clock signals when this clock signal is high (more positive/less negative voltage). Used as a qualifier (example: clock on both edges of K only if J is high).

CLOCK SIGNALS	The **r, f, x, l,** and **h** operators may be used on the following clock signals: **J, K, L, M** or **N.**

Defaults

If no parameters are specified, the current clock definitions are displayed. After powerup or **tinit**, the **u** option is always set. Other clock options set at initialization depend on the particular emulator in use and whether or not there is an external analyzer present.

Examples

To display the current settings of the master clocks after powerup or a **tinit**, type:

```
M> tck
```

You will see:

```
tck -r L -ub -s S
```

Here, the emulation analyzer is set to clock both user and background code into the analyzer on the rising edge of L; the clock speed is less than 16 MHz.

To trace user code on the falling edge of **L** when **M** is high, at a speed between 16 and 20 MHz, type:

```
M> tck -u -f L -h M -s F
```

```
!ERROR 1239 : tck - clock speed not available with current count qualifier
```

Since the clock speed is fast, we cannot count time with **tcq**. Let's disable **tcq** and re-execute the command. Type:

```
M> tcq none
M> tck -u -f L -h M -s F
```

Now the command completed successfully. Verify the settings by typing:

```
M> tck
```

You will see:

```
tck -f L -h M -u -s F
```

Now let's add tracing of background code to the current clock settings.

```
M> tck -ub
```

Verify the changes by typing:

```
M> tck
```

You will see:

```
tck -f L -ub -s F
```

You'll note that the M clock qualifier was removed. If you modify any of the qualifier, speed, or edge parameters, you must re-specify the entire configuration for that particular parameter. The rest of the trace clock specification is left alone. (In the example above, we modified the qualifiers by changing **-u** to **-ub**; part of the original qualifier spec was **-h M**. To retain this, we would have to specify **-ub -h M**.)

Let's modify the external analyzer clocks. Type:

```
M> xtmo -s
```

This defines the external analyzer as an independent state analyzer. Now type:

```
M> xtck -r J -h K -s VF
```

Here, we've set the external analyzer to clock on the rising edge of **J**, but only when **K** is high. The clock rate is set to a rate between 20 and 25 MHz.

Verify the setting by typing:

```
M> xtck
```

You will see:

```
tck -r J -h K -s VF
```

Related Commands

ta (display current trace signal activity. This can be useful after you have modified the clocks for the external analyzer; you can issue a **ta** command and verify that you are seeing activity on the signals of interest.)

tcq (used to specify trace count qualifier for states, time, or none; maximum clock speed set in **tck** affect which **tcq** parameters are valid)

tsck (used to define slave clock signals used by the analyzer; **tck** defines the master clock signals. Default mode for **tsck** is off on all pods.)

xtv (specifies threshold voltages for external analyzer input lines; must be set correctly to ensure that the **J** and **K** clock signals are recognized)

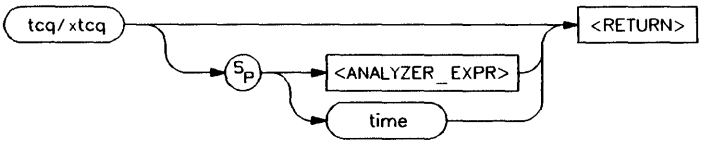
xtmo (specifies mode of operation for the external analyzer; that is, whether it acts as an independent analyzer or is appended to the emulation analyzer)

tcq,xtcq

Summary

Specify analyzer trace tag count qualifier

Syntax



Function

The **tcq** (**xtcq**) command allows you to specify a qualifier for the emulation (external) trace tag counter.

When the tag counter is active, the analyzer counts occurrences of the expression you specify (which may include simple or complex expressions (depending on analyzer configuration), **time**, or **none**). Each time a trace state is stored, the value of the counter is also stored and the counter is reset. The tag counter shares trace memory with stored states, so only half as many states can be captured by the analyzer when the tag counter is active. (The analyzer can store 1024 states with **tcq none**, 512 states otherwise.)

Parameters

<ANALYZER_EXPR>

<ANALYZER_EXPR> allows you to specify an expression to be counted by the trace tag counter. This expression consists of a **<SIMPLE_EXPR>** in analyzer easy configuration and a **<COMPLEX_EXPR>** in complex configuration. Refer to the syntax pages for expressions for specific details of analyzer expressions. In either configuration, the expression may consist of the states **any** (count all states) or **none** (disable trace tag counting).

Note 

The count qualifier **tcq arm** is not permitted in any configuration.

time — If you specify **time** rather than an analyzer expression, the trace tag counter measures the amount of time between stored states.

Note 

The **tcq time** qualifier is only available when the analyzer clock speed is set to the slow (**S**) speed setting (default). If the clock speed is set to very fast (**VF**), then trace tag counting must be turned off by specifying **tcq none**. Refer to the **tck** command (analyzer clock specification) for further information.

Defaults

If no parameters are given, the current count qualifier is displayed. Upon powerup or after **tinit** initialization, the clock qualifier defaults to the state **tcq time**.

Examples

If you want to view the current **tcq** setting, type:

```
M> tcq
```

You will see:

```
tcq time
```

To see the effects of counting no states, you can set up the following measurement (this measurement uses the 68000 sample program in Appendix A):

```
M> tg addr=2000
M> t
M> tcq none
```

```
Emulation trace started
```

```
U> r 2000
U> th

Emulation trace halted
```

Now, view the trace listing from the above measurement by typing:

```
U> tl -d
```

```
Line   addr,H  68000 Mnemonic                          count,R  seq
-----  ------  ----------------------------------  ---------  ---
    0  002000  MOVEA.L 0001000,A2                  *********   +
    1  002002    0000  supr prog                   *********   .
    2  002004    1000  supr prog                   *********   .
    3  002006  MOVEA.L 0001004,A3                  *********   .
    4  001000    0000  supr data rd word           *********   .
    5  001002    3000  supr data rd word           *********   .
    6  002008    0000  supr prog                   *********   .
    7  00200a    1004  supr prog                   *********   .
    8  00200c  MOVE.B  #000,[A2]                   *********   .
    9  001004    0000  supr data rd word           *********   .
```

Note the asterisks in the **count** field; no states were counted. To count time intervals, set up the following measurement:

```
U> tcq time
U> tg

tg addr=2000

U> t

Emulation trace started

U> r 2000
U> th

Emulation trace halted

U> tl -d
```

```
 Line   addr,H  68000 Mnemonic                          count,R  seq
-----  ------  ----------------------------------  ---------  ---
   -1  000004    2000  supr data rd word                 ---   .
    0  002000  MOVEA.L 0001000,A2                  0.400 uS   +
    1  002002    0000  supr prog                   0.400 uS   .
    2  002004    1000  supr prog                   0.400 uS   .
    3  002006  MOVEA.L 0001004,A3                  0.400 uS   .
    4  001000    0000  supr data rd word           0.400 uS   .
    5  001002    3000  supr data rd word           0.400 uS   .
    6  002008    0000  supr prog                   0.400 uS   .
    7  00200a    1004  supr prog                   0.400 uS   .
    8  00200c  MOVE.B  #000,[A2]                   0.400 uS   .
```

Here, the relative amount of time measured between storage states is 0.400 uS. You can change the trace listing so that the time intervals are displayed as an absolute value relative to the trigger state instead of the last state stored. Type:

```
U> tf addr,h mne count,a seq
U> tl -td
```

(We used the **-t** option to **tl** to ensure the top states of the trace are displayed).

```
 Line   addr,H  68000 Mnemonic                          count,A  seq
-----   ------  ---------------------------------  ---------  ---
   -1   000004    2000  supr data rd word          -0.400 uS   .
    0   002000  MOVEA.L 0001000,A2                       0      +
    1   002002    0000  supr prog                    0.400 uS   .
    2   002004    1000  supr prog                    0.800 uS   .
    3   002006  MOVEA.L 0001004,A3                   1.200 uS   .
    4   001000    0000  supr data rd word            1.600 uS   .
    5   001002    3000  supr data rd word            2.000 uS   .
    6   002008    0000  supr prog                    2.400 uS   .
    7   00200a    1004  supr prog                    2.800 uS   .
    8   00200c  MOVE.B  #000,[A2]                    3.200 uS   .
```

Note that the time interval is now measured relative to the trigger state. Let's reset the trace format to count relative:

```
U> tf addr,h mne count,r seq
```

You may want to count the number of accesses to the input pointer address of 3000 hex. To make such a measurement, type:

```
U> tsto addr!=3000
U> tcq addr=3000
U> t

Emulation trace started

U> r 2000
U> th

Emulation trace halted
```

```
U> tl -td 30
```

Line	addr,H	68000 Mnemonic	count,R	seq
-1	000004	2000 supr data rd word	---	.
0	002000	MOVEA.L 0001000,A2	0	+
1	002002	0000 supr prog	0	.
2	002004	1000 supr prog	0	.
3	002006	MOVEA.L 0001004,A3	0	.
4	001000	0000 supr data rd word	0	.
5	001002	3000 supr data rd word	0	.
6	002008	0000 supr prog	0	.
7	00200a	1004 supr prog	0	.
8	00200c	MOVE.B #000,[A2]	0	.
9	001004	0000 supr data rd word	0	.
10	001006	4000 supr data rd word	0	.
11	00200e	0000 supr prog	0	.
12	002010	MOVE.B [A2],D0	0	.
13	003000	00 supr data wr byte	1	.
14	002012	CMPI.B #000,D0	0	.
15	003000	00 supr data rd byte	1	.
16	002014	0000 supr prog	0	.
17	002016	BEQ.B 0002010	0	.
18	002018	CMPI.B #**,D0	0	.
19	002010	MOVE.B [A2],D0	0	.
20	002012	CMPI.B #000,D0	0	.
21	003000	00 supr data rd byte	1	.
22	002014	0000 supr prog	0	.
23	002016	BEQ.B 0002010	0	.
24	002018	CMPI.B #**,D0	0	.
25	002010	MOVE.B [A2],D0	0	.
26	002012	CMPI.B #000,D0	0	.
27	003000	00 supr data rd byte	1	.
28	002014	0000 supr prog	0	.

In the above listing, we've seen four different accesses to the address specified. Using the **tf** command, you can specify that these accesses are to be displayed as a count relative to the trigger state (as with counting time). Type:

```
U> tf addr,h mne count,a seq
U> tl -td 30
```

```
 Line    addr,H  68000 Mnemonic                          count,A  seq
-----    ------  ---------------------------------      ---------  ---
   -1    000004    2000  supr data rd word                      0   .
    0    002000  MOVEA.L 0001000,A2                           0   +
    1    002002    0000  supr prog                            0   .
    2    002004    1000  supr prog                            0   .
    3    002006  MOVEA.L 0001004,A3                           0   .
    4    001000    0000  supr data rd word                    0   .
    5    001002    3000  supr data rd word                    0   .
    6    002008    0000  supr prog                            0   .
    7    00200a    1004  supr prog                            0   .
    8    00200c  MOVE.B  #000,[A2]                            0   .
    9    001004    0000  supr data rd word                    0   .
   10    001006    4000  supr data rd word                    0   .
   11    00200e    0000  supr prog                            0   .
   12    002010  MOVE.B  [A2],D0                              0   .
   13    003000      00  supr data wr byte                    1   .
   14    002012  CMPI.B  #000,D0                              1   .
   15    003000      00  supr data rd byte                    2   .
   16    002014    0000  supr prog                            2   .
   17    002016  BEQ.B   0002010                              2   .
   18    002018  CMPI.B  #**,D0                              2   .
   19    002010  MOVE.B  [A2],D0                              2   .
   20    002012  CMPI.B  #000,D0                              2   .
   21    003000      00  supr data rd byte                    3   .
   22    002014    0000  supr prog                            3   .
   23    002016  BEQ.B   0002010                              3   .
   24    002018  CMPI.B  #**,D0                              3   .
   25    002010  MOVE.B  [A2],D0                              3   .
   26    002012  CMPI.B  #000,D0                              3   .
   27    003000      00  supr data rd byte                    4   .
   28    002014    0000  supr prog                            4   .
```

Again, four accesses to address 3000 hex were recorded; however, the occurrences are now displayed relative to the trigger state.

You can set up more complex count patterns in complex configuration. For example, with the 68000 program, you might wish to count occurrences of the CLEAR_LOOP routine without storing the states associated with the routine. To make this measurement, first set the analyzer to complex configuration by typing:

```
M> tcf -c
```

You next set up the patterns to use in the complex expressions. You'll need a pattern to trigger on; make this the address of the OUTPUT routine. You will also need a data value of 00 to count only nulls written to the output area. In addition, you will need two range values. Since the analyzer only has one range variable, you can roughly approximate a second range variable by using don't care values in a pattern expression.

Set up the patterns and range expression as follows:

```
M> tpat p1 addr=2052
M> tpat p5 data=00
M> tpat p6 addr!=10000001xxxxxxy
M> trng addr=4000..40ff
```

Now specify the count qualifier. You want to count all states where the address range 4000 through 40ff is accessed when data is equal to zero. Type:

```
M> tcq r and p5
```

However, you don't want to store any of the states associated with the clear routine. Type:

```
M> tsto 2 !r and p6
```

To set up the sequencer to trigger on term 2 after the address 2052 is encountered, type:

```
M> tif 1 p1 2
M> tif 2 never
M> tsq -t 2
```

Now, begin the measurement by typing:

```
M> t
```

```
Emulation trace started
```

```
M> r 2000
U> m 3000=41
```

(If you don't modify the input area, the output routine will never execute; thus, the analyzer will not find its trigger and no measurement will occur.)

```
U> th
```

```
Emulation trace halted
```

```
U> tf addr,h mne count,R seq
U> tl -td 10
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
    0    002052  MOVEA.L A3,A1                            ---   +
    1    001008       54  supr data rd byte               32   .
    2    001009       48  supr data rd byte                0   .
    3    00100a       49  supr data rd byte                0   .
    4    00100b       53  supr data rd byte                0   .
    5    00100c       20  supr data rd byte                0   .
    6    00100d       49  supr data rd byte                0   .
    7    00100e       53  supr data rd byte                0   .
    8    00100f       20  supr data rd byte                0   .
    9    001010       4d  supr data rd byte                0   .
```

Looking at line number 1 of the trace listing, you will see that the counter has a value of 32. Thus, the condition where address was in the range 4000 through 40ff hex with data equal to zero occurred 32 times. Notice that the counter resets on the next state, since no further occurrences of the count pattern were found.

Related Commands

tck (used to specify the clock source and clock parameters for the analyzer)

tp (specifies position of the trigger within the trace; note that **tcq** affects the number of states the analyzer can store and therefore may affect trigger positioning)

tpat (assigns analyzer expressions to pattern names in complex configuration; the pattern names are then used to specify qualifiers in other analyzer commands such as **tcq**)

trng (specifies a range of values to be used as a complex mode qualifier; this range definition can be used as a count qualifier by **tcq**)

tsq (used to manipulate the trace sequencer)

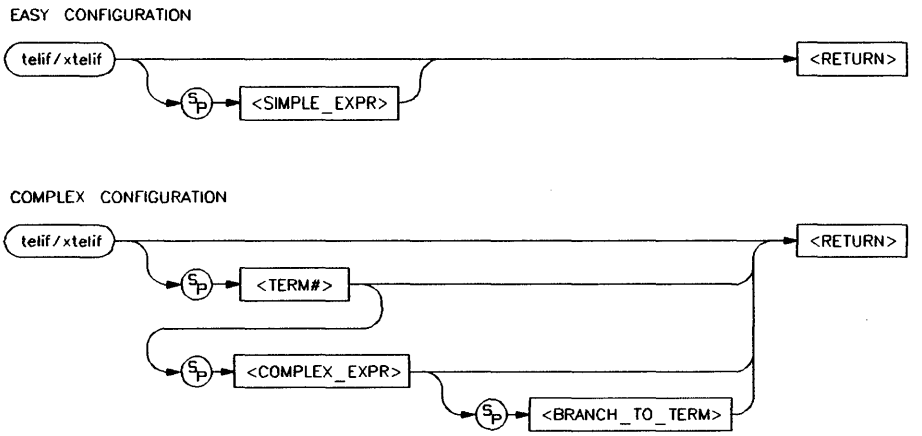
xtmo (used to choose the external analyzer mode; the external analyzer can operate as an independent state or timing analyzer, or it may be appended to the emulation analyzer. If appended, the **xtcq** command has no effect and the **tcq** command specifies the count qualifier for both analyzers.)

telif,xtelif

Summary

Specify sequencer secondary branch qualifier

Syntax



Function

The **telif** (**xtelif**) command allows you to set the global restart qualifier (in easy configuration) for the emulation (external) analyzer sequencer. In complex configuration, **telif** (**xtelif**) lets you set the secondary branch qualifier for each term of the emulation (external) analyzer sequencer.

Note

The **telif** command is used as a global restart qualifier in easy configuration and a secondary branch qualifier in complex configuration. The hierarchy of the **tif** and **telif** commands is such that either branch will be taken if found before the other; however, if both branches are found simultaneously, the **tif** branch is always taken over the **telif** branch.

When in easy configuration, the sequencer will restart by jumping to sequencer term number one (1) when the expression specified by **telif** occurs.

When in complex configuration, the sequencer will branch to the sequencer level specified by the **<BRANCH_TO_TERM>** parameter when the expression specified is found. There are always eight sequencer terms available. Position of the trigger term is defined with the **tsq** command. If both the **tif** and **telif** expressions are satisfied simultaneously, the **tif** branch is taken; otherwise, branching occurs according to which expression is first satisfied.

Note 

If the **tif** expression for the given **<TERM#>** has a **<COUNT>** parameter other than one (1), the counter is reset to zero (0) if the **telif** branch is taken before the occurrence counter parameter is satisfied. For example, if the **tif** counter parameter is 7, and the **tif** expression has been found 5 times, then the **telif** expression is satisfied, the **telif** branch will be taken and the **tif** counter will be reset from 5 to 0. This might cause you difficulty if you happen to have **telif** branching back to the same term; your occurrence condition may or may not be satisfied.

Parameters (Easy Configuration)

<SIMPLE_EXPR> **<SIMPLE_EXPR>** lets you directly specify an analyzer expression to use as a global restart qualifier. For example, **<SIMPLE_EXPR>** might consist of the expression **addr=2000**. For detailed information on specification of simple expressions, refer to the expression syntax pages.

Parameters (Complex Configuration)

<TERM#>

<TERM#> lets you specify a sequencer term number to associate with the given **<COMPLEX_EXPR>**. When you associate a term number with a complex expression, that expression is only used as a secondary branch qualifier at the sequencer level specified by the term number. If you specify **<TERM#>** without an expression, the secondary branch qualifier currently associated with that term number is displayed.

<COMPLEX_EXPR>

<COMPLEX_EXPR> allows you to specify complicated analyzer expressions made up of relationships between simple analyzer expressions. When you create a complex expression, you must first assign pattern names (**p1-p8**) to simple expressions using the **tpat** command. You then use the pattern names and relational operators to create complex expressions. For example, if you wish to branch from term 1 to term 2 when **address=2000** and **data=20** or when **address=2000** and **data=42**, you would use the following commands:

```
U> tpat p1 addr=2000 and data=20
U> tpat p2 addr=2000 and data=42
U> telif 1 p1 | p2 2
```

The | symbol represents an intra-set OR operator. For more information on complex expressions, operators, and pattern sets, refer to the expression syntax pages within this manual.

<BRANCH_TO_TERM>

The **<BRANCH_TO_TERM>** parameter allows you to indicate the branch destination when the **<COMPLEX_EXPR>** is found. For example, you may wish to have the sequencer branch from term 1 to term 3 after the expression is found. This would be specified as **telif 1 <COMPLEX_EXPR> 3**. If you do not specify a term number, the default is to increment the sequencer level (**telif <TERM#> <COMPLEX_EXPR> (<TERM#> + 1)**).

Defaults

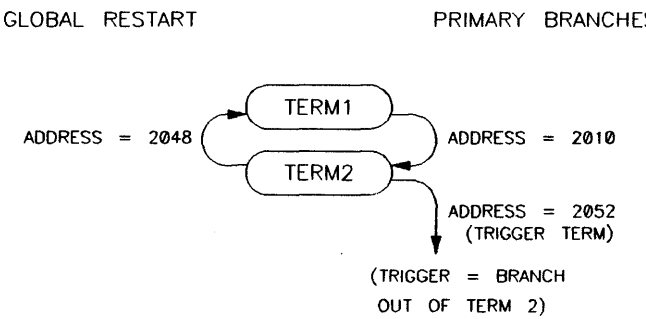
If **telif** is entered with no parameters, the global restart qualifier or secondary branch qualifiers (depending on analyzer configuration) for all sequencer levels are displayed. If **telif** is entered with only a **<TERM#>** parameter in complex configuration, the secondary branch qualifier for only that term number is displayed.

Upon initialization via a powerup sequence or the **tinit** command, the secondary branch specifiers are set to **telif never**.

In complex configuration, if **<BRANCH_TO_TERM>** is not specified, the default is (**<TERM#> + 1**).

Note

At sequencer term number 8, the default branch to condition is **<TERM#>**; that is, branch to the same term.



Examples

When the analyzer is in easy configuration, the **telif** command allows you to specify a global restart qualifier. This means that the analyzer will restart the sequencer when the qualifier is satisfied. To illustrate this, you can create an example using the 68000 sample program from Appendix A. In this example, the analyzer first looks for the READ_INPUT routine, then the OUTPUT routine. However, if anything other than 41 hex ("A") or 42 hex ("B") is input to the input area, the analyzer will restart; that is, it will ignore the OUTPUT routine and begin searching again for the READ_INPUT routine. In this manner, you can have the analyzer record the OUTPUT routine only when an "A" or "B" is input. To create the example, type:

```
U> tsq -i 2
```

This command inserts an extra sequence term. (Remember that the analyzer is initialized in easy configuration with a one term sequencer; you will need two terms for this measurement.) Now you must set up the branch qualifiers. Type:

```
U> tif 1 addr=2010
U> tif 2 addr=2052
```

You can use the UNRECOGNIZED set up routine as the qualifier for restarting the analyzer. Type:

```
U> telif addr=2048
```

Now make the measurement:

```
U> t

Emulation trace started

U> r 2000
```

To illustrate the restart, modify the input area value to an unrecognized command. Type:

```
U> m 3000=43
```

Now check the trace status:

```
U> ts
```

You will see:

```
--- Emulation Trace Status ---
NEW User trace running
Arm ignored
Trigger not in memory
Arm to trigger ?
States ? (512) ?..?
Sequence term 2
Occurrence left 1
```

Note that the trigger was not found. Another way to view this is as follows:

```
U> tl
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ----------------------------------  ---------  ---
** Trigger not in memory **
```

Even though the analyzer saw the value 2052 hex, it was told to ignore it by the **telif** command instructing it to return and search for 2010 hex. Now you can try satisfying the analyzer conditions. Type:

```
U> m 3000=41
U> ts
```

```
--- Emulation Trace Status ---
NEW User trace complete
Arm ignored
Trigger in memory
Arm to trigger ?
States 512 (512) -1..510
Sequence term 3
Occurrence left 1
```

```
U> tl -d
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ----------------------------------  ---------  ---
   -1    002038  ORI.B   #**,[A2]+                         ---   .
    0    002052  MOVEA.L A3,A1                        0.600 uS   +
    1    002054  MOVE.B  #020,D1                      0.400 uS   .
    2    002056    0020  supr prog                    0.400 uS   .
    3    002058  MOVEA.L A3,A5                        0.400 uS   .
    4    00205a  MOVE.B  #000,[A5]+                   0.400 uS   .
    5    00205c    0000  supr prog                    0.400 uS   .
    6    00205e  SUBI.W  #00001,D1                    0.400 uS   .
    7    004000      00  supr data wr byte            0.400 uS   .
    8    002060    0001  supr prog                    0.400 uS   .
```

In this instance, the address 2048 was never encountered by the analyzer, so the sequencer was not restarted. Therefore, the trigger was found and the trace completed.

In complex configuration, the **telif** command allows you to branch to any sequence term from any other term. The example below is based on a problem found in an incorrectly loaded version of the program from Appendix A. Apparently, when commands "A" or "B" are entered, the program does not execute the output routine.

Let's insert a bug in the program to simulate this problem. Type:

```
U> m -db 2039=34
```

You can set up a measurement to trace the problem; although only one **telif** command is used as a restart branch, it illustrates the branching possible. First, initialize the analyzer and set it to complex configuration by typing:

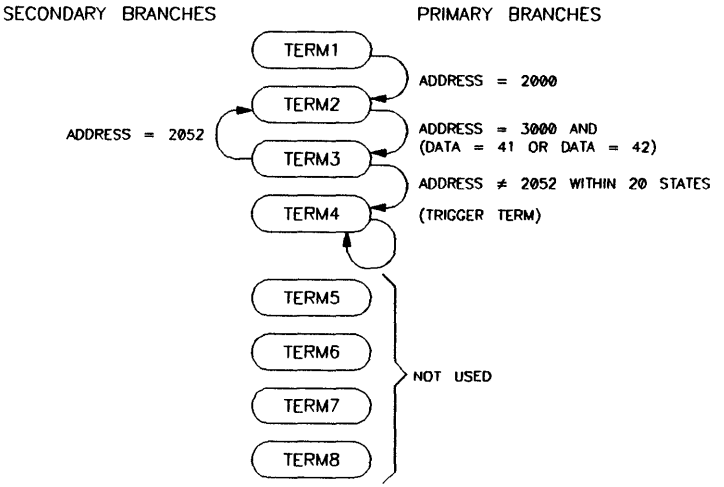
```
U> tinit
U> tcf -c
```

Now you can set up the patterns necessary. We want a pattern to start the sequencer which indicates that the program has begun executing, so we'll just use the starting address of 2000 hex. We want to recognize that a command "A" or "B" has been read, so we will set up patterns for the address and command values. (Note the use of the newly defined lowerdata label; this is set up because we don't know what resides at location 3001 hex and don't want to bother finding out.) Finally, we define patterns for OUTPUT and NOT OUTPUT addresses.

```
U> tpat p1 addr=2000
U> tpat p2 addr=3000
U> tpat p3 addr=2052
U> tpat p4 addr!=2052
U> tlb lowerdata 40..47
U> tpat p5 lowerdata=41
U> tpat p6 lowerdata=42
```

Now you can set up the branch conditions. By issuing the commands below, you will specify that the sequencer transitions from term 1 to term 2 when the program start address is recognized. Next, the sequencer will transition from term 2 to term 3 when either a command "A" or command "B" is read from the

input area. Finally, if the OUTPUT routine is not recognized within 20 states, the analyzer triggers. If OUTPUT is recognized before 20 additional states occur, no problem was found and the sequencer branches back to term 2 (this is the **telif** branch), looking for another command input. Note that the trigger position is set to the end of the trace; this allows you to see the events leading up to the trigger.




```
U> tif 1 p1 2
U> tif 2 p2 and p5|p6 3
U> tif 3 p4 4 20
U> tif 4 never
U> tsq -t 4
U> telif 3 p3 2
U> tp e
```

Now you can begin the measurement. Type:

```
U> t
```

```
Emulation trace started
```

```
U> r 2000
U> m 3000=41
```

(This command effectively enters a command "A" into the input area.)

```
                                U> ts

--- Emulation Trace Status ---
  NEW User trace complete
  Arm ignored
  Trigger in memory
  Arm to trigger ?
  States 1024 (1024) -1022..1
  Sequence term 4
  Occurrence left 1

                                U> tl -d -20..0

   Line    addr,H  68000 Mnemonic                          count,R  seq
  -----    ------  --------------------------------  ---------  ---
    -20    003000       41  supr data rd byte          0.400 uS   +
    -19    002014  ORI.B    #0f8,D0                   0.400 uS   .
    -18    002016    67f8  supr prog                  0.400 uS   .
    -17    002018  CMPI.B   #041,D0                   0.400 uS   .
    -16    00201a    0041  supr prog                  0.800 uS   .
    -15    00201c  BEQ.W    000202c                   0.400 uS   .
    -14    00201e    000e  supr prog                  0.400 uS   .
    -13    00202c  MOVE.B   #011,D0                   0.600 uS   .
    -12    00202e    0011  supr prog                  0.400 uS   .
    -11    002030  MOVEA.L  #000001008,A0             0.400 uS   .
    -10    002032    0000  supr prog                  0.400 uS   .
     -9    002034    1008  supr prog                  0.400 uS   .
     -8    002036  BRA.W    000206c                   0.400 uS   .
     -7    002038    0034  supr prog                  0.400 uS   .
     -6    00206c  JMP      000200c                   0.600 uS   .
     -5    00206e    0000  supr prog                  0.400 uS   .
     -4    002070    200c  supr prog                  0.400 uS   .
     -3    00200c  MOVE.B   #000,[A2]                 0.400 uS   .
     -2    00200e    0000  supr prog                  0.400 uS   .
     -1    002010  MOVE.B   [A2],D0                   0.400 uS   .
      0    003000      00  supr data wr byte          0.400 uS   +
```

The trigger was found, indicating that after command "A" was read (line -20), the OUTPUT routine was not accessed within 20 states. If you look at the listing, you will notice that at state -6, the program jumped to address 206c and then back to 200c; therefore, it avoided the output routine entirely. To track down the problem, you can use the memory mnemonic display capability on the COMMAND_A routine:

```
                U> m -dm 202c..204e

  00202c  103c0011     MOVE.B  #011,D0
  002030  207c000010   MOVEA.L #000001008,A0
  002036  60000034     BRA.W   000206C
  00203a  103c0011     MOVE.B  #011,D0
  00203e  207c000010   MOVEA.L #000001019,A0
  002044  6000000c     BRA.W   0002052
  002048  103c000f     MOVE.B  #00F,D0
  00204c  207c000010   MOVEA.L #00000102A,A0
```

Notice that the branch instruction at 2036 jumps to 206c. The problem is the relative branch value in location 2039 hex; it should be 1a rather than 34. You can repair this using the following command:

```
U> m -db 2039=1a
```

See the **tsq** syntax pages for further examples of complex configuration **telif** commands.

Related Commands

tarm (allows you to specify that the **trig1** or **trig2** signal will arm the analyzer. This arm condition can then be used as part of the secondary branch qualifier)

tcf (used to select whether the analyzer is operated in easy configuration or complex configuration)

tif (used to specify a primary branch specification for the analyzer)

tg (used to set up a simple trigger qualifier in either analyzer configuration. Specifying the **tg** command overrides the current sequencer specification and will modify the existing **telif** qualifier stored in sequence term number 1)

tpat (used to assign pattern names to simple expressions for use in specifying complex expressions. These complex expressions are used to specify **telif** qualifiers in analyzer complex configuration)

trng (used to set up an expression which assigns a range of values to a range variable. This range information may be used in specifying complex **telif** qualifiers)

tsto (specifies a global trace storage qualifier in both easy & complex configurations; also specifies a trace storage qualifier for each sequencer term in complex configuration. Used to control the types of information stored by the analyzer)

tsq (used to manipulate the trace sequencer)

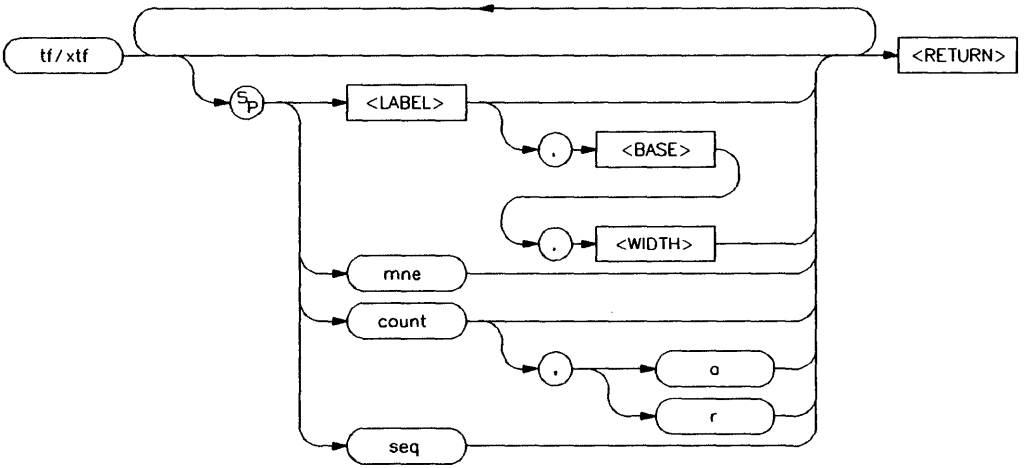
xtmo (specifies whether the external analyzer operates as an independent state or timing analyzer or is appended to the emulation analyzer. If appended to the emulation analyzer, the **xtelif** command is invalid; all secondary branch qualifiers are specified with the **telif** command)

tf,xtf

Summary

Specify the trace list display format

Syntax



Function

The **tf** (**xtf**) command allows you to specify which pieces of information from the emulation (external) analyzer trace will be displayed by **tl** (**xtl**) (trace list) commands.

Each format item specifies a column of the trace list display. See "Parameters" for a list of the possible format items.

Note

Changing the trace format DOES NOT change the type of information captured by the analyzer; it only specifies how the captured data should be displayed.

Parameters

<LABEL>

If you specify **<LABEL>**, the analyzer bits associated with that label will be displayed in a column of the trace listing with **<LABEL>** as the column header.

<BASE>

<BASE> allows you to specify the numeric base in which <LABEL> is to display. The choices are **Y** (binary), **Q** or **O** (octal), **T** (decimal), **H** (hexadecimal), or **A** (ASCII). The specifiers are not case sensitive. In ASCII mode, non printing characters are displayed as periods (.). If **<BASE>** is not specified, the default base is hexadecimal.

<WIDTH>

This option allows you to set the width of the address field to values from 4 to 50. If your emulator supports symbols, by setting **<WIDTH>**, you can view symbols in the address field when you display memory mnemonic.

<LABEL>, **<BASE>**, and **<WIDTH>** must each be separated by a comma (,).

mne

If you specify **mne**, the disassembled mnemonic for each instruction captured by the analyzer is displayed. To ensure correct operation of **mne**, the labels **addr, data, stat** and **extra** (if applicable) must be defined according to their power up defaults for the target processor being emulated; otherwise, incorrect disassembly may occur. The **mne** format item is only allowed with the **tf** command, and not with **xtf**.

count

If you specify **count**, the state or tag time counter defined by **tcq** is displayed in the trace list. If you have designated prestore states via the **tpq** command, these prestore

states will be flagged in the **count** column of the trace list.

a — Specifying **count,a** causes the state/time counter to display the count in absolute mode; that is, each counter value is shown relative to the trigger state. Therefore, states before the trigger will show as negative values and states after the trigger will show as positive values. Prestore states do not have counts.

r — Specifying **count,r** causes the state/time counter to display the count in relative mode; that is, each counter value is shown relative to the previous state. As with **count,a**, prestore states do not have counts.

seq — If you specify **seq**, an indicator is printed for each state which caused the sequencer to branch from one term to another (whether the same term or a different term).

Defaults

If no parameters are given, the current settings of the trace format are displayed. Upon powerup or after a **tinit** command, the trace format is **tf addr,H mne count,R seq**. (This command is for the 68000 emulator; other HP 64700-Series emulators may have slightly different **tf** definitions upon initialization.)

Note

Various **tf** format items may be concatenated as desired on the command line by including a space between each format item.

Examples

The examples below were created using the 68000 sample program in Appendix A.

To view the default trace format, type:

```
M> tf
```

```
tf addr,H mne count,R seq
```

With this specification, there will be four information columns in the trace list. The first will be address in hexadecimal; the second is disassembled processor mnemonics, the third is the state/time counter value, and the fourth is the trace sequencer status. To see the resulting trace list, you can trace part of the sample program.

```
M> tg addr=2000
M> t
```

```
Emulation trace started
```

```
M> r 2000
U> tl -d
```

You will see:

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
   -1    000004    2000  supr data rd word                  ---   .
    0    002000  MOVEA.L 0001000,A2                    0.400 uS   +
    1    002002    0000  supr prog                     0.400 uS   .
    2    002004    1000  supr prog                     0.400 uS   .
    3    002006  MOVEA.L 0001004,A3                    0.400 uS   .
    4    001000    0000  supr data rd word             0.400 uS   .
    5    001002    3000  supr data rd word             0.400 uS   .
    6    002008    0000  supr prog                     0.400 uS   .
    7    00200a    1004  supr prog                     0.400 uS   .
    8    00200c  MOVE.B  #000,[A2]                     0.400 uS   .
```

Perhaps all the information you need is the address and data values in hexadecimal. To put the trace list in that format, type:

```
U> tf addr,H data,H
U> tl -td
```

You will see:

```
 Line  addr,H  data,H
-----  ------  ------
   -1  000004  2000
    0  002000  2479
    1  002002  0000
    2  002004  1000
    3  002006  2679
    4  001000  0000
    5  001002  3000
    6  002008  0000
    7  00200a  1004
    8  00200c  14bc
```

Or maybe you would rather have the address in decimal format and the data in binary. Type:

```
U> tf addr,T data,Y
U> tl -td
```

You will see:

```
 Line  addr,T    data,Y
-----  --------  ----------------
   -1  00000004  0010000000000000
    0  00008192  0010010001111001
    1  00008194  0000000000000000
    2  00008196  0001000000000000
    3  00008198  0010011001111001
    4  00004096  0000000000000000
    5  00004098  0011000000000000
    6  00008200  0000000000000000
    7  00008202  0001000000000100
    8  00008204  0001010010111100
```

The processor status information is not part of the default trace format. To display this information in binary, type:

```
U> tf addr,H mne stat,Y
U> tl -td
```

You will see:

```
 Line   addr,H  68000 Mnemonic                      stat,Y
-----   ------  ----------------------------------  --------
   -1   000004    2000  supr data rd word           11101110
    0   002000  MOVEA.L 0001000,A2                  11110110
    1   002002    0000  supr prog                   11110110
    2   002004    1000  supr prog                   11110110
    3   002006  MOVEA.L 0001004,A3                  11110110
    4   001000    0000  supr data rd word           11101110
    5   001002    3000  supr data rd word           11101110
    6   002008    0000  supr prog                   11110110
    7   00200a    1004  supr prog                   11110110
    8   00200c  MOVE.B  #000,[A2]                   11110110
```

When you define labels for groups of analyzer input lines using the **tlb** command, you can use these labels in trace format specifications. Suppose you are interested in seeing what types of ASCII information are transferred on the lower byte of the data bus. Type:

```
U> tlb lowerdata 40..47
```

Now you can use this label in the trace format. Type:

```
U> tf addr,H mne lowerdata,A
U> tl -td
```

You will see:

```
 Line    addr,H  68000 Mnemonic                          lowerdata,A
-----    ------  --------------------------------------  -----------
   -1    000004    2000  supr data rd word               .
    0    002000  MOVEA.L 0001000,A2                      $
    1    002002    0000  supr prog                       .
    2    002004    1000  supr prog                       .
    3    002006  MOVEA.L 0001004,A3                      &
    4    001000    0000  supr data rd word               .
    5    001002    3000  supr data rd word               0
    6    002008    0000  supr prog                       .
    7    00200a    1004  supr prog                       .
    8    00200c  MOVE.B  #000,[A2]                       .
```

You can display the trace sequencer information also. To do this along with a status display in hex, type:

```
U> tf addr,H mne stat,H seq
U> tl -td
```

You will see:

```
 Line    addr,H  68000 Mnemonic                          stat,H  seq
-----    ------  --------------------------------------  ------  ---
   -1    000004    2000  supr data rd word               ee       .
    0    002000  MOVEA.L 0001000,A2                      f6       +
    1    002002    0000  supr prog                       f6       .
    2    002004    1000  supr prog                       f6       .
    3    002006  MOVEA.L 0001004,A3                      f6       .
    4    001000    0000  supr data rd word               ee       .
    5    001002    3000  supr data rd word               ee       .
    6    002008    0000  supr prog                       f6       .
    7    00200a    1004  supr prog                       f6       .
    8    00200c  MOVE.B  #000,[A2]                       f6       .
```

In addition, state/time counter information can be displayed. Type:

```
U> tf addr,H mne count seq
```

(The counter display defaults to count absolute unless specified otherwise.)

```
U> tl -td
```

You will see:

```
 Line    addr,H  68000 Mnemonic                          count,A  seq
-----    ------  --------------------------------  ---------  ---
   -1    000004    2000  supr data rd word          -0.400 uS   .
    0    002000  MOVEA.L 0001000,A2                       0      +
    1    002002    0000  supr prog                    0.400 uS   .
    2    002004    1000  supr prog                    0.800 uS   .
    3    002006  MOVEA.L 0001004,A3                    1.200 uS   .
    4    001000    0000  supr data rd word            1.600 uS   .
    5    001002    3000  supr data rd word            2.000 uS   .
    6    002008    0000  supr prog                    2.400 uS   .
    7    00200a    1004  supr prog                    2.800 uS   .
    8    00200c  MOVE.B  #000,[A2]                    3.200 uS   .
```

To change the counter display to count relative, type:

```
U> tf addr,H mne count,R seq
U> tl -td 0..28
```

You will see:

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  --------------------------------  ---------  ---
   -1    000004    2000  supr data rd word              ---     .
    0    002000  MOVEA.L 0001000,A2                    0.400 uS   +
    1    002002    0000  supr prog                    0.400 uS   .
    2    002004    1000  supr prog                    0.400 uS   .
    3    002006  MOVEA.L 0001004,A3                    0.400 uS   .
    4    001000    0000  supr data rd word            0.400 uS   .
    5    001002    3000  supr data rd word            0.400 uS   .
    6    002008    0000  supr prog                    0.400 uS   .
    7    00200a    1004  supr prog                    0.400 uS   .
    8    00200c  MOVE.B  #000,[A2]                    0.400 uS   .
    9    001004    0000  supr data rd word            0.400 uS   .
   10    001006    4000  supr data rd word            0.400 uS   .
   11    00200e    0000  supr prog                    0.400 uS   .
   12    002010  MOVE.B  [A2],D0                      0.400 uS   .
   13    003000      00  supr data wr byte            0.400 uS   .
   14    002012  CMPI.B  #000,D0                      0.400 uS   .
   15    003000      00  supr data rd byte            0.400 uS   .
   16    002014    0000  supr prog                    0.400 uS   .
   17    002016  BEQ.B   0002010                      0.400 uS   .
   18    002018  CMPI.B  #**,D0                       0.400 uS   .
   19    002010  MOVE.B  [A2],D0                      0.600 uS   .
   20    002012  CMPI.B  #000,D0                      0.400 uS   .
   21    003000      00  supr data rd byte            0.400 uS   .
   22    002014    0000  supr prog                    0.400 uS   .
   23    002016  BEQ.B   0002010                      0.400 uS   .
   24    002018  CMPI.B  #**,D0                       0.400 uS   .
   25    002010  MOVE.B  [A2],D0                      0.600 uS   .
   26    002012  CMPI.B  #000,D0                      0.400 uS   .
   27    003000      00  supr data rd byte            0.400 uS   .
   28    002014    0000  supr prog                    0.400 uS   .
```

Notice above the recurring CMPI.B instruction at location 2018, even though the branch at location 2016 is constantly taken back to location 2010. The CMPI.B instruction is showing up because the processor is prefetching this instruction; the HP 64700 emulator for the 68000 cannot determine whether or not prefetched instructions have actually been executed.

Related Commands

tl,xtl (displays the current data in emulation (external) trace memory according to the specifications set up by **tf**)

tlb,xtlb (define labels which represent groups of emulation (external) analyzer input lines; these labels may be used to create special trace list displays by including the labels in the **tf** definition)

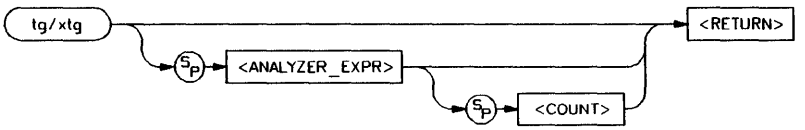
xtmo (defines whether the external analyzer acts as an independent state/timing analyzer or is appended to the emulation analyzer)

tg,xtg

Summary

Specify a trigger condition for the analyzer

Syntax



Function

The **tg** (**xtg**) command sets a trigger condition for the emulation (external) analyzer.

When the expression specified occurs the number of times specified in the **<COUNT>** parameter, the analyzer has found its trigger.

The **tg** command modifies the current analyzer sequence specification. (Refer to the **tsq** command description for further information regarding the trace sequencer.) The manner in which the sequencer is modified is dependent upon the analyzer configuration.

If the analyzer is in easy configuration (**tcf -e**), the sequencer is reduced to a one term sequence triggering upon exit from term 1. The global restart qualifier is set to never (**telif never**); the primary branch condition is set to the specified trigger expression (**tif 1 <EXPR> <COUNT>**).

If the analyzer is in complex configuration (**tcf -c**), the sequencer is modified to trigger upon entrance to the second sequence term (**tsq -t 2**), the secondary branch qualifier is set to never (**telif 2 never**), and the primary branch qualifier for term number 1 is set to the specified expression (**tif 1 <EXPR> 2 <COUNT>**).

The analyzer storage qualifier (**tsto**) is not affected in either configuration; therefore, the analyzer uses the storage qualifier from the most recent **tsto** command.

Parameters

<ANALYZER_ EXPR>	**<ANALYZER_EXPR>** allows you to specify the expression to recognize as a trigger. This expression consists of a **<SIMPLE_EXPR>** in analyzer easy configuration and a **<COMPLEX_EXPR>** when the analyzer is in complex configuration. Refer to the syntax pages for expressions for specific details of analyzer expressions. In either configuration, the expression may consist of the states **any** or **all** (trigger on any state) or **none** or **never** (don't trigger the analyzer).
<COUNT>	You use the **<COUNT>** parameter to specify the number of times the expression **<ANALYZER_EXPR>** must occur before the trigger condition is satisfied. **<COUNT>** is specified as a decimal integer value; if **<COUNT>** is not specified, the default is one (1).

Defaults

If no parameters are specified, the current primary branch condition for sequencer term 1 is displayed. Note that this is not necessarily the trigger condition, depending on the analyzer commands leading up to this point. (For more help on this concept, refer to the examples below, other trace command descriptions, and the *Analyzer User's Guide*.) After powerup or **tinit** initialization, **tg** is set to **tg any**.

Examples

These examples were created using the sample 68000 program in Appendix A.

When operating the analyzer in easy configuration, using the **tg** command resets the sequencer to a two term sequence with a primary branch in term number one corresponding to the trigger condition. For example, you may have been working with the following analyzer sequence:

```
U> tsq
```

You will see:

```
tif 1 addr=2010
tif 2 addr=2052
tsto all
telif addr=2048
```

Here, there are two sequence terms, with conditional branches from term 1 to term 2 and from term 2 to the trigger. There is also a global restart specification. Now, if you want to trigger the analyzer on the 32nd occurrence of address 205A, type:

```
U> tg addr=205a 32
```

To see the new sequencer specification, type:

```
U> tsq
```

```
tif 1 addr=205a 32
tsto all
telif never
```

To proceed with the measurement, type:

```
U> t
```

```
Emulation trace started
```

```
U> r 2000
U> m 3000=41
```

(This command inputs a "command" value for the program; otherwise, the program will never execute the routine at 205a hex.)

```
U> tl -d
```

Line	addr,H	68000 Mnemonic		count,R	seq
0	00205a	MOVE.B	#000,[A5]+	---	+
1	00205c	0000	supr prog	0.400 uS	.
2	00205e	SUBI.W	#00001,D1	0.400 uS	.
3	00401f	00	supr data wr byte	0.400 uS	.
4	002060	0001	supr prog	0.400 uS	.
5	002062	BNE.B	000205a	0.400 uS	.
6	002064	MOVE.B	[A0]+,[A1]+	0.400 uS	.
7	002066	SUBI.W	#00001,D0	0.800 uS	.
8	001008	54	supr data rd byte	0.400 uS	.
9	004000	54	supr data wr byte	0.400 uS	.

The trigger condition was found; line 0 shows the 32nd iteration of the trigger value.

In analyzer complex configuration, a similar situation exists. The **tg** command defines simple sequence specification and overwrites sequencer terms 1 and 2 to create the new specification. For example, assume you were working with the following sequencer definition:

```
U> tsq
```

```
tif 1 p1 2
tif 2 p2 and p5 | p6 3
tif 3 p4 4 20
tif 4 never
tif 5 any 6
tif 6 any 7
tif 7 any 8
tif 8 never
tsq -t 4
tsto 1 all
tsto 2 all
tsto 3 all
tsto 4 all
tsto 5 all
tsto 6 all
tsto 7 all
tsto 8 all
telif 1 never
telif 2 never
telif 3 p3 2
telif 4 never
telif 5 never
telif 6 never
telif 7 never
telif 8 never
```

Notice that the trigger term is in term number 4; terms 1 through 3 have various branch qualifiers specified which determine how the sequencer advances to term 4. Assume that you've fixed the problem you were troubleshooting with this sequence specification; now you just want to trigger the analyzer on any command. First, you may need an extra pattern definition. Type:

```
U> tpat
```

```
tpat p1 addr=2000
tpat p2 addr=3000
tpat p3 addr=2052
tpat p4 addr!=2052
tpat p5 lowerdata=41
tpat p6 lowerdata=42
tpat p7 any
tpat p8 any
```

To trigger on any command (even unrecognized ones) you will need a pattern where data is not equal to zero. Type:

```
U> tlb lowerdata 40..47
U> tpat p7 lowerdata!=00
```

Now you can define the trigger. Type:

```
U> tg p2 and p7
```

Note 

An occurrence count can also be specified for complex configuration triggers; this particular example does not illustrate a count.

You might want to see how the sequencer has been modified. Type:

```
U> tsq
```

```
tif 1 p2 and p7 2
tif 2 never
tif 3 p4 4 20
tif 4 never
tif 5 any 6
tif 6 any 7
tif 7 any 8
tif 8 never
tsq -t 2
tsto 1 all
tsto 2 all
tsto 3 all
tsto 4 all
tsto 5 all
tsto 6 all
tsto 7 all
tsto 8 all
telif 1 never
telif 2 never
telif 3 p3 2
telif 4 never
telif 5 never
telif 6 never
telif 7 never
telif 8 never
```

Notice that term 2 has been redefined as the trigger term (**tsq -t 2** and **tif 2 never**). This was done automatically by the **tg** command. Sequencer term number 1 has been redefined with the new trigger branch qualifier of **tif 1 p2 and p7 2**.

To proceed with the measurement, type:

```
U> t
```

```
Emulation trace started
```

```
U> r 2000
U> m 3000=23
```

(This command inputs an "unrecognized" command to the program.)

```
U> tl -d 21
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  --------------------------------  ---------  ---
    0    003000       41   supr data rd byte              ---   +
    1    002014  ORI.B    #0f8,D0                   0.400 uS   .
    2    002016    67f8   supr prog                  0.400 uS   .
    3    002018  CMPI.B   #041,D0                   0.400 uS   .
    4    00201a    0041   supr prog                  0.800 uS   .
    5    00201c  BEQ.W    000202c                   0.400 uS   .
    6    00201e    000e   supr prog                  0.400 uS   .
    7    00202c  MOVE.B   #011,D0                   0.600 uS   .
    8    00202e    0011   supr prog                  0.400 uS   .
    9    002030  MOVEA.L  #000001008,A0             0.400 uS   .
   10    002032    0000   supr prog                  0.400 uS   .
   11    002034    1008   supr prog                  0.400 uS   .
   12    002036  BRA.W    0002052                   0.400 uS   .
   13    002038    001a   supr prog                  0.400 uS   .
   14    002052  MOVEA.L  A3,A1                     0.600 uS   .
   15    002054  MOVE.B   #020,D1                   0.400 uS   .
   16    002056    0020   supr prog                  0.400 uS   .
   17    002058  MOVEA.L  A3,A5                     0.400 uS   .
   18    00205a  MOVE.B   #000,[A5]+                0.400 uS   .
   19    00205c    0000   supr prog                  0.400 uS   .
   20    00205e  SUBI.W   #00001,D1                 0.400 uS   .
```

At line number zero (0) of the trace listing, you can see that the analyzer did in fact trigger on address 3000 with data not equal to zero.

Related Commands

bc (allows you to break the emulator to the monitor when various conditions occur; you can have the emulator break upon analyzer trigger by specifying **tgout trig1** and **bc -e trig1** (or you could use the trig2 signal to perform the same function))

t (starts an emulation trace)

tarm (used to specify an analyzer arm condition; the analyzer will not trigger until the arm condition is received if you specify **tg arm**)

tcf (used to specify whether the analyzer is operated in easy or complex configuration)

tpat (used to assign pattern names to simple analyzer expressions; the pattern names are then used in creating complex analyzer expressions which could be used with the **tg** command to trigger the analyzer)

trng (used to specify a range of values for a particular group of analyzer lines; this range may be used in specifying complex analyzer expressions for triggering the analyzer)

tsto (specifies which states encountered by the analyzer should be stored in trace memory)

tsq (used to manipulate the trace sequencer. Note that the sequencer's current status is affected by the **tg** command.)

xtmo (specifies whether the external analyzer is treated as a separate state or timing analyzer or is appended to the emulation analyzer. If appended to the emulation analyzer, the **xtg** command is no longer valid; **tg** sets the trigger condition for both analyzers.)

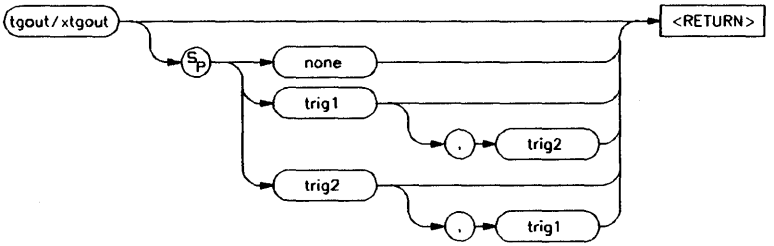
Notes

tgout,xtgout

Summary

Specify signals to drive upon analyzer trigger

Syntax



Function

The **tgout** (**xtgout**) command allows you to specify which of the internal trig1 and/or trig2 signals will be driven when the emulation analyzer (external analyzer) finds its trigger condition.

Note that if the analyzer is receiving trig1 or trig 2 via the **tarm** command, then that signal cannot be driven, although no error message will be issued to that effect.

If the external analyzer has been appended to the emulation analyzer via the **xtmo** command, then the **xtgout** command is invalid and the **tgout** command specifies the trigger signals to be driven when either analyzer finds its trigger.

Parameters

none	If **none** is specified, neither the trig1 nor trig2 signals are driven when the analyzer finds its trigger state.
trig1	If **trig1** is specified, then the trig1 signal is driven by the analyzer when the trigger state is found.

trig2

If **trig2** is specified, then the trig2 signal is driven by the analyzer when the trigger state is found.

To specify that both trig1 and trig2 should be driven when the analyzer trigger is found, concatenate both options with a comma: **tgout trig1,trig2**.

Defaults

If no parameters are specified, the current state of **tgout** is displayed. Upon powerup or **tinit**, the default state is **tgout none**.

Examples

You may wish to have the emulator break to monitor execution upon receipt of the analyzer trigger. Type the following:

```
M> tcf -e
M> bc -e trig1
M> tgout trig1
M> tg addr=2000
M> r
```

The emulator will break to the emulation monitor when it encounters the trigger state of address 2000 hexadecimal.

To display the state of **tgout** after powerup, type:

```
M> tgout
```

You will see:

```
tgout none
```

Related Commands

bc (allows you to specify a break to emulation monitor when the tgout condition is satisfied)

bnct (specifies whether or not trig1 and trig2 are used to drive and/or receive the rear panel BNC connector signal line)

cmbt (specifies whether or not trig1 and trig2 are used to drive and/or receive the CMB trigger signal)

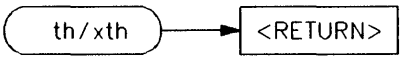
tarm (used to specify that the analyzer will be armed upon assertion or negation of trig1 or trig2)

th,xth

Summary

Halt the analyzer

Syntax



Function

The **th** (**xth**) command stops an emulation (external) trace.

If the external analyzer has been appended to the emulation analyzer with the **xtmo** command, the **xth** command is invalid and **th** halts both the emulation and external trace in process.

The analyzer will stop driving the **trig1** and **trig2** signals when the trace is halted. This may cause you difficulty in making measurements with instruments connected to the BNC. For example, if you set the HP 64700 analyzer to drive **trig1** (**tgout trig1**) when the trigger condition is found, then pipe this to the BNC connector with **bnct -d trig1**, the BNC signal will be driven high when the HP 64700 analyzer finds its trigger while a trace is in progress; it will fall low when the trace finishes.

You should start the HP 64700 trace after you have begun the external instrument's measurement. Otherwise, the following measurement errors may occur, depending on the type of external instrument you are using:

- With an edge sensitive instrument, starting the instrument after the HP 64700 finds the analyzer trigger will mean that the instrument never sees the transition of the **trig1** line and therefore never triggers.
- With a level sensitive instrument, starting the instrument after the HP 64700 finds the trigger will mean that the instrument triggers immediately; although many states of interest have probably already passed.

Note

If the analyzer trigger specification has not been found, you will need to use the **th** command to halt the analyzer before you can display the trace list.

Parameters

None.

Defaults

Does not apply.

Examples

To start an emulation trace, type:

```
M> t
```

You will see:

```
Emulation trace started
```

To stop the emulation trace, type:

```
M> th
```

You will see:

```
Emulation trace halted
```

Related Commands

t (used to start an analyzer trace)

ts (allows you to determine the current status of the emulation analyzer)

tx (starts an analyzer trace upon receipt of the CMB execute signal)

x (starts a synchronous CMB execution)

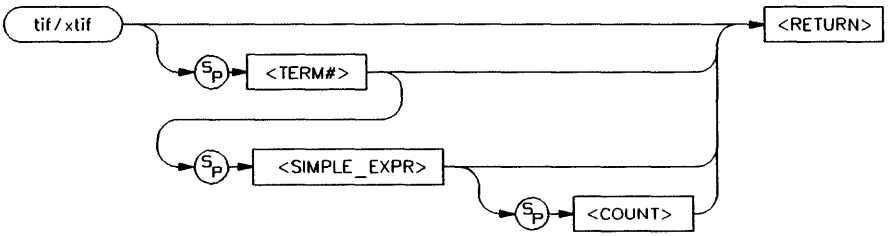
tif,xtif

Summary

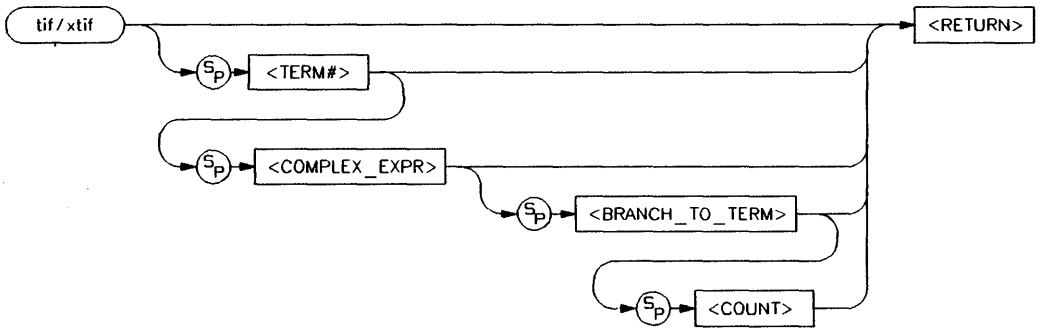
Specify sequencer primary branch qualifiers

Syntax

EASY CONFIGURATION



COMPLEX CONFIGURATION



Function

The **tif** (**xtif**) command allows you to set the primary branch qualifier for each term of the emulation (external) analyzer sequencer.

Note

The **telif** command is used as a global restart qualifier in easy configuration and a secondary branch qualifier in complex configuration. The hierarchy of the **tif** and **telif** commands is such that either branch will be taken if found before the other; however, if both branches are found simultaneously, the **tif** branch is always taken over the **telif** branch.

When in easy configuration, the sequencer will increment to the next sequencer level when the expression specified by **tif** occurs the number of times specified by the **<COUNT>** parameter. There is a maximum of four sequence levels; only one is available at initialization. If you require more sequencer levels, you must insert them with the **tsq** command. (The term you are specifying a primary branch for with the **tif** command must be present in the sequence.) The branch out of the last sequencer term constitutes the trigger.

When in complex configuration, the sequencer will branch to the sequencer level specified by the **<BRANCH_TO_TERM>** parameter when the expression specified occurs the number of times indicated in the **<COUNT>** parameter. There are always eight sequencer terms available. Position of the trigger term is defined with the **tsq** command.

Parameters (Easy configuration)

<TERM#> — When you specify **<TERM#>**, it indicates which sequencer term's primary branch qualifier is to be modified with the qualifier specified in the **<SIMPLE_EXPR>** parameter. If you specify **<TERM#>** without an expression, the **tif** qualifier for that term number is displayed.

<SIMPLE_EXPR> **<SIMPLE_EXPR>** lets you directly specify an analyzer expression to use as a storage qualifier. For example, **<SIMPLE_EXPR>** might consist of the expression **addr=2000**. For detailed information on specification of simple expressions, refer to the expression syntax pages.

<COUNT> You use the **<COUNT>** parameter to specify the number of times the expression **<SIMPLE_EXPR>** must occur before the primary branch condition is satisfied. **<COUNT>** is specified as a decimal integer value; if **<COUNT>** is not specified, the default is one (1).

Parameters (Complex configuration)

<TERM#> **<TERM#>** lets you specify a sequencer term number to associate with the given **<COMPLEX_EXPR>**. When you associate a term number with a complex expression, that expression is used as a branch qualifier at the sequencer level specified by the term number. If you specify **<TERM#>** without an expression, the complex expression currently associated with that term number is displayed.

<COMPLEX_ EXPR> **<COMPLEX_EXPR>** allows you to specify complicated analyzer expressions made up of relationships between simple analyzer expressions. When you create a complex expression, you must first assign pattern names (**p1-p8**) to simple expressions using the **tpat** command. You then use the pattern names and relational operators to create complex expressions. For example, if you wish to branch from term 1 to term 2 when

address=2000 and **data=20** or when **address=2000** and **data=42**, you would use the following commands:

```
U> tpat p1 addr=2000 and data=20
U> tpat p2 addr=2000 and data=42
U> tif 1 p1 | p2 2
```

The | symbol represents an intra-set OR operator. For more information on complex expressions, operators, and pattern sets, refer to the expression syntax pages within this manual.

<BRANCH_ TO_TERM>

The **<BRANCH_TO_TERM>** parameter allows you to indicate the branch destination when the **<COMPLEX_EXPR>** is found. For example, you may wish to have the sequencer branch from term 1 to term 3 after the expression is found. This would be specified as **tif 1 <COMPLEX_EXPR> 3**. If you do not specify a term number, the default is to increment the sequencer level (**tif <TERM#> <COMPLEX_EXPR> (<TERM#> + 1)**).

<COUNT>

You use the **<COUNT>** parameter to specify the number of times the expression **<COMPLEX_EXPR>** must occur before the primary branch condition is satisfied. **<COUNT>** is specified as a decimal integer value; if **<COUNT>** is not specified, the default is one (1).

Note

If you specify the **<COUNT>** parameter, you must also specify a **<BRANCH_TO_TERM>** parameter. If you omit the **<BRANCH_TO_TERM>** parameter when specifying **<COUNT>**, the system will interpret the count as "branch to term" information; if greater than eight (8), an error will be returned; otherwise, you will have just specified an incorrect branch.

Defaults

If **tif** is entered with no parameters, the primary branch qualifiers for all sequencer levels are displayed. If **tif** is entered with only a **<TERM#>** parameter, the primary branch qualifier for only that term number is displayed.

Upon initialization via a powerup sequence or the **tinit** command, the primary branch specifiers are set to **tif <TERM#> any (<TERM#> + 1)**.

In complex configuration, if **<BRANCH_TO_TERM>** is not specified, the default is **(<TERM#> + 1)**; if **<COUNT>** is not specified, the default count is one (1).

Note

At sequencer term number 8, the default branch to condition is **<TERM#>**; that is, branch to the same term.

Examples

Given the example 68000 program from appendix A, you might want to trigger the analyzer when INIT and CLEAR are executed, CLEARLOOP is executed 32 times, and LOOP is executed the number of times necessary to move message A or message B. Further, you would like to position the trigger so that you can see all of the states recorded up until the trigger state.

You can set up some equates of values to labels for use in the trace qualifiers. Type:

```
U> equ init=2000
U> equ clear=200c
U> equ clearloop=205a
```

```
U> equ loop=2064
U> equ clearcount=32T
U> equ abloop=17T
U> equ unrecloop=0f
```

Now you will need to insert additional sequencer terms. (The analyzer defaults to one sequence term in the easy configuration after initialization via **tinit** or powerup.) Type:

```
U> tsq -i 2
U> tsq -i 3
U> tsq -i 4
```

Next, set up the sequencer branch conditions. With the commands given below, the analyzer will first look for the INIT routine, then the CLEAR routine, then it will look for 32 occurrences of the CLEARLOOP routine and 17 occurrences of the LOOP routine before triggering. Type:

```
U> tif 1 addr=init
U> tif 2 addr=clear
U> tif 2 addr=clearloop clearcount
U> tif 4 addr=loop abloop
```

You can view the sequencer modifications by typing:

```
U> tsq
```

```
tif 1 addr=init
tif 2 addr=clear
tif 3 addr=clearloop 32
tif 4 addr=loop 17
tsto all
telif never
```

To position the trigger state at the end of the trace memory, type:

```
U> tp e
```

Now you can start the measurement. Type:

```
M> t
```

```
Emulation trace started
```

```
M> r 2000
U> m 3000=41
```

With the **m** command, you have effectively input a "command" "A" to the program. Now look at the trace listing:

```
U> tl -d -20..0
```

This command lists the last 21 states of the trace. You will see:

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  --------------------------------------  --------  ---
  -20    002066  SUBI.W  #00001,D0                        0.400 uS   .
  -19    001015      47  supr data rd byte                0.400 uS   .
  -18    00400d      47  supr data wr byte                0.400 uS   .
  -17    002068    0001  supr prog                        0.400 uS   .
  -16    00206a  BNE.B   0002064                          0.400 uS   .
  -15    00206c  JMP     ********                         0.400 uS   .
  -14    002064  MOVE.B  [A0]+,[A1]+                      0.600 uS   .
  -13    002066  SUBI.W  #00001,D0                        0.400 uS   .
  -12    001016      45  supr data rd byte                0.400 uS   .
  -11    00400e      45  supr data wr byte                0.400 uS   .
  -10    002068    0001  supr prog                        0.400 uS   .
   -9    00206a  BNE.B   0002064                          0.400 uS   .
   -8    00206c  JMP     ********                         0.400 uS   .
   -7    002064  MOVE.B  [A0]+,[A1]+                      0.600 uS   .
   -6    002066  SUBI.W  #00001,D0                        0.400 uS   .
   -5    001017      20  supr data rd byte                0.400 uS   .
   -4    00400f      20  supr data wr byte                0.400 uS   .
   -3    002068    0001  supr prog                        0.400 uS   .
   -2    00206a  BNE.B   0002064                          0.400 uS   .
   -1    00206c  JMP     ********                         0.400 uS   .
    0    002064  MOVE.B  [A0]+,[A1]+                      0.600 uS   +
```

The trigger condition is the last state; this is where the analyzer has found the 17th occurrence of the LOOP routine.

In complex configuration, you can set up even more complex qualifiers. These can be used to store only execution between certain addresses (windowing) until the trace memory is full. For example, if you want to store only execution in the output routines for each command of the sample program, use the following procedure.

First, initialize the analyzer and set it to complex configuration. Type:

```
M> tinit
M> tcf -c
```

We will need to define an analyzer label for the lower 8 data bits so we can correctly qualify byte wide data transfers. Type:

```
M> tlb lowerdata 40..47
```

Now we can define the analyzer patterns necessary to set up the sequencer and storage qualifiers. Type:

```
M> trng addr=4000..4011
M> tpat p1 addr=200c
M> tpat p2 addr=3000
M> tpat p3 addr=2018
M> tpat p4 addr=2064
M> tpat p5 lowerdata!=00
M> tpat p6 addr=202c
M> tpat p7 addr=203a
M> tpat p8 addr=2048
```

Next, you need to define the sequencer pattern. With the commands below, the sequencer will jump from term 1 to term 2 when the CLEAR routine is found. It will jump from term 2 to term 3 when a command is read (that is, data not equal to zero) from address 3000 hex. Next, it will jump from term 3 to term 4 when the PROCESS_COMM routine is found. The next jump is from term 4 to term 5 when any of the command setup routines (COMMAND_A,COMMAND_B, or UNRECOGNIZED) is found. An increment from term 5 to term 6 occurs when the LOOP routine is found. Finally, the sequencer will jump from term 6 back to term 2 when the CLEAR routine is found in subsequent passes.

You may wonder why the branch from term 6 is identical to that of term 1. This is done so that different store specifications can be made. The first time the CLEAR routine is found, it is stored. On subsequent passes, the CLEAR routine address is not stored.

To set up the sequencer branches, type the following commands:

```
M> tif 1 p1 2
M> tif 2 p2 and p5 3
M> tif 3 p3 4
M> tif 4 p6|p7|p8 5
M> tif 5 p4 6
M> tif 6 p1 2
```

Set the trigger term at term number 2 by typing:

```
M> tsq -t 2
```

Now you can set up the storage conditions. Type:

```
M> tsto 1 p1
M> tsto 2 p2 and p5
M> tsto 3 p3
M> tsto 4 p6|p7|p8
M> tsto 5 p4
M> tsto 6 r and p5
```

To see the messages written to the output area, alter the trace listing format as follows:

```
M> tf addr,H mne lowerdata,A seq
```

Now you can begin the measurement. Type:

```
M> t
```

```
Emulation trace started
```

```
M> r 2000
```

The next three commands effectively "input" three different commands to the program. The first two are "command A" and "command B"; the third is unrecognized.

```
U> m 3000=41
U> m 3000=42
U> m 3000=43
```

Now view the resultant trace list by typing:

```
U>  tl -d 0..63
```

This displays the first 64 trace states. You will see:

Line	addr,H	68000 Mnemonic	lowerdata,A	seq
0	00200c	MOVE.B #**,[A2]	.	+
1	003000	41 supr data rd byte	A	+
2	002018	CMPI.B #**,D0	.	+
3	00202c	MOVE.B #**,D0	.	+
4	002064	MOVE.B [A0]+,[A1]+	.	+
5	004000	54 supr data wr byte	T	.
6	004001	48 supr data wr byte	H	.
7	004002	49 supr data wr byte	I	.
8	004003	53 supr data wr byte	S	.
9	004004	20 supr data wr byte	.	.
10	004005	49 supr data wr byte	I	.
11	004006	53 supr data wr byte	S	.
12	004007	20 supr data wr byte	.	.
13	004008	4d supr data wr byte	M	.
14	004009	45 supr data wr byte	E	.
15	00400a	53 supr data wr byte	S	.
16	00400b	53 supr data wr byte	S	.
17	00400c	41 supr data wr byte	A	.
18	00400d	47 supr data wr byte	G	.
19	00400e	45 supr data wr byte	E	.
20	00400f	20 supr data wr byte	.	.
21	004010	41 supr data wr byte	A	.
22	00200c	MOVE.B #**,[A2]	.	+
23	003000	42 supr data rd byte	B	+
24	002018	CMPI.B #**,D0	.	+
25	00203a	MOVE.B #**,D0	.	+
26	002064	MOVE.B [A0]+,[A1]+	.	+
27	004000	54 supr data wr byte	T	.
28	004001	48 supr data wr byte	H	.
29	004002	49 supr data wr byte	I	.
30	004003	53 supr data wr byte	S	.
31	004004	20 supr data wr byte	.	.
32	004005	49 supr data wr byte	I	.
33	004006	53 supr data wr byte	S	.
34	004007	20 supr data wr byte	.	.
35	004008	4d supr data wr byte	M	.
36	004009	45 supr data wr byte	E	.
37	00400a	53 supr data wr byte	S	.
38	00400b	53 supr data wr byte	S	.
39	00400c	41 supr data wr byte	A	.
40	00400d	47 supr data wr byte	G	.
41	00400e	45 supr data wr byte	E	.
42	00400f	20 supr data wr byte	.	.
43	004010	42 supr data wr byte	B	.
44	00200c	MOVE.B #**,[A2]	.	+
45	003000	43 supr data rd byte	C	+
46	002018	CMPI.B #**,D0	.	+
47	002048	MOVE.B #**,D0	.	+
48	002064	MOVE.B [A0]+,[A1]+	.	+
49	004000	49 supr data wr byte	I	.
50	004001	4e supr data wr byte	N	.
51	004002	56 supr data wr byte	V	.
52	004003	41 supr data wr byte	A	.
53	004004	4c supr data wr byte	L	.
54	004005	49 supr data wr byte	I	.
55	004006	44 supr data wr byte	D	.
56	004007	20 supr data wr byte	.	.

```
57    004008      43  supr data wr byte          C           .
58    004009      4f  supr data wr byte          O           .
59    00400a      4d  supr data wr byte          M           .
60    00400b      4d  supr data wr byte          M           .
61    00400c      41  supr data wr byte          A           .
62    00400d      4e  supr data wr byte          N           .
63    00400e      44  supr data wr byte          D           .
```

If you look at the **lowerdata** column, you will see commands "A", "B", and "C" entered, and the respective output messages. The analyzer would continue to store states every time a new command is entered until trace memory is filled; then it will stop storing new states.

Related Commands

tarm (allows you to specify that the **trig1** or **trig2** signal will arm the analyzer. This arm condition can then be used as part of the primary branch qualifier)

tcf (used to select whether the analyzer is operated in easy configuration or complex configuration)

telif (used to specify a secondary branch specification for the analyzer)

tg (used to set up a simple trigger qualifier in either analyzer mode. Specifying the **tg** command overrides the current sequencer specification and will modify the existing **tif** qualifier stored in sequence term number 1)

tpat (used to assign pattern names to simple expressions for use in specifying complex expressions. These complex expressions are used to specify **tif** qualifiers in analyzer complex configuration)

trng (used to set up an expression which assigns a range of values to a range variable. This range information may be used in specifying complex **tif** qualifiers)

tsto (specifies a global trace storage qualifier in both easy and complex configurations; also specifies a trace storage qualifier for each sequencer term in complex configuration. Used to control the types of information stored by the analyzer)

tsq (used to manipulate the trace sequencer)

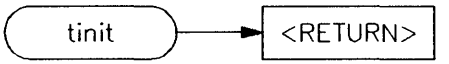
xtmo (specifies whether the external analyzer operates as an independent state or timing analyzer or is appended to the emulation analyzer. If appended to the emulation analyzer, the **xtif** command is invalid; all primary branch qualifiers are specified with the **tif** command)

tinit

Summary

Reset trace specification

Syntax



Function

The **tinit** command restores all trace specification items to their powerup default values. See "Defaults."

Parameters

None.

Defaults

These are the powerup defaults for the trace specification:

Analyzer arm

```
tarm always
```

Trace Configuration

```
tcf -e
```

Note that if the trace configuration was complex, it is reset to easy configuration.

Analyzer master clocks

```
tck -r L -u -s S
```

The analyzer clock configuration at powerup is dependent on the particular HP 64700-Series Emulator in use.

Trace count qualifier

```
tcq time
```

Trace format

```
tf addr,H mne count,R seq
```

The trace format may vary depending on the particular emulator in use.

Trace trigger

```
tg any
tgout none
```

Analyzer signal line labels

```
#### Emulation trace labels
tlb addr 0..23
tlb data 32..47
tlb stat 24..31
```

These labels will vary according to the emulator in use.

Trigger Position

```
tp s
```

Trace Prestore Qualifier

```
tpq none
```

Trace sequencer (includes branch and store conditions)

```
tif 1 any
tsto all
telif never
```

Trace slave clocks

```
tsck -o 1
tsck -o 2
tsck -o 3
```

Trace Upon Execute?

```
tx -d    # ignore the execute signal
```

Examples

To reset the analyzer parameters to the powerup defaults, type:

```
M> tinit
```

When the M> prompt returns, the analyzer has been re-initialized.

Related Commands

init (used to initialize selected portions of the emulator or the entire emulator, dependent on the options given)

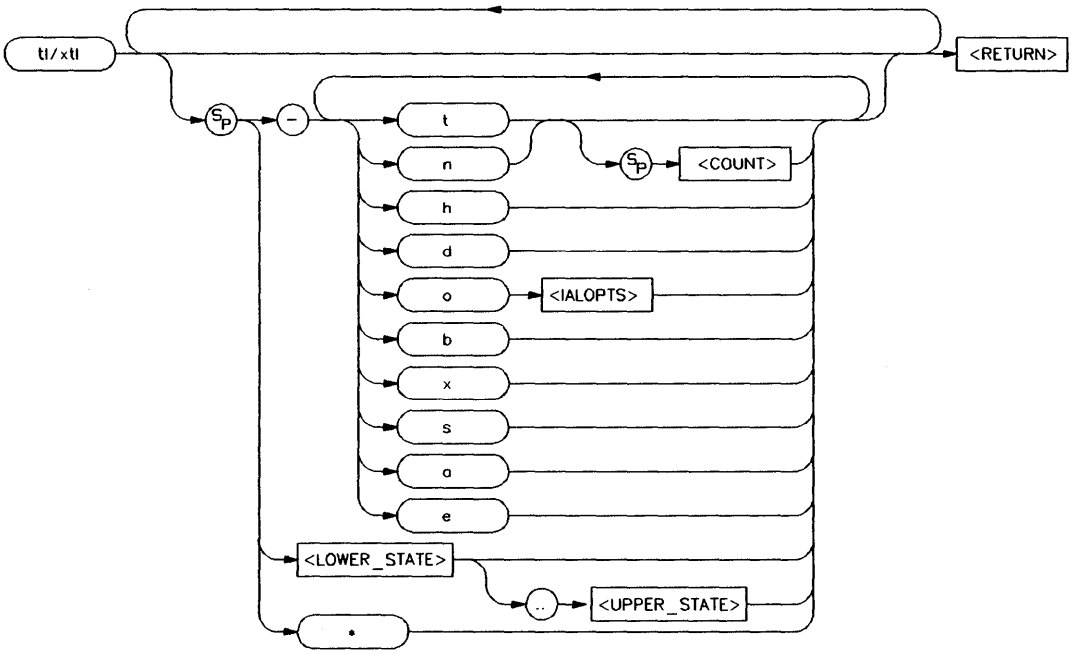
Notes

tl, xtl

Summary

Display/dump current trace list

Syntax



Function

The **tl** (**xtl**) command allows you to display the current emulation (external) analyzer trace list information. Options are available which allow you to specify disassembly of instructions, number of states to display and starting state to display.

You may also dump the trace list to a host computer using the **-b** (binary) or **-x** (hexadecimal) options in conjunction with the HP 64000 **transfer** software. This allows you to perform post processing of the trace data on your host. See Appendix B of this manual for details on the binary and hexadecimal trace list formats.

If the trigger specification has not yet been satisfied, the trace list cannot be displayed until the trace in progress is halted with the **th** command. Entering the **tl** command before the trace is halted results in the message "**** Trigger not in memory ****."

If the analyzer was halted before any states were captured, the message "**** No trace data ****" is displayed upon entry of the **tl** command.

Parameters

t	Display the top states of the trace. If you have specified the number of states to display with the **<COUNT>** parameter, that number of states is displayed. Otherwise, the default is to display the same number of states as the last time **tl** was invoked to display part (but not all) of the trace.
n	Display the next states of the trace. If you have specified the number of states to display with the **<COUNT>** parameter, that number of states is displayed. Otherwise, the same number of states will be displayed as the last time you used **tl** to display part (but not all) of the trace.
<COUNT>	**<COUNT>** allows you to specify the number of states to display with the **-t** or **-n** options.
h	Normally, column headers are displayed at the top of each trace list. These label the state number, count, and each trace field specified by the **tf** command. Specifying the **-h** option allows you to suppress printing of the column headers.
d	Some emulators do not disassemble instruction data in the trace list automatically (the 68000 emulator falls into

this category). Specifying the **-d** option results in disassembly of instructions starting with the first state to be displayed. Other emulators may ignore this option.

o — Certain emulators allow specific inverse assembler options for trace list instruction disassembly. By specifying **-o** and **<IALOPTS>**, you can control disassembly of the trace list. Refer to your *Emulator User's Guide* for specific information on the **<IALOPTS>** supported by your emulator. Some emulators, such as the 68000, do not support this option.

b — The **-b** option dumps the trace list in binary format using the HP 64000 **transfer** protocol. Refer to Appendix B of this manual for details on the binary trace list format.

x — The **-x** option dumps the trace list in hexadecimal format using the HP 64000 **transfer** protocol. Refer to Appendix B of this manual for details on the hexadecimal trace list format.

Note 

The **-h**, **-d**, and **-o** options cannot be used with either **-b** or **-x**. Also, the **-b** and **-x** options cannot be used together.

---

s — This allows you to display symbols in the address column.

a — This allows you to display absolute addresses in the address column. This is the default.

e — This allows you to display symbols and absolute addresses in the address column.

Note

The HP 64700 remembers the last option specified for the address field (**-s**, **-a**, or **-e**), and uses it for the next **tl** command if no other option is specified.

*	If you specify *****, the entire trace list is displayed. Notice that **tl** does not recognize displaying the entire trace as the last default count. (This helps avoid filling your screen with lots of trace list data on subsequent **tl** commands.)
<LOWER_ STATE>	If you specify **<LOWER_STATE>**, the trace display starts with that state.
<UPPER_ STATE>	If you specify both **<LOWER_STATE>** and **<UPPER_STATE>**, the trace list contains all states between the lower and upper state inclusive.

Note

If you specify a lower state, it must be done without using the **-t** or **-n** options, as the Terminal Interface will interpret your lower state specification as a **<COUNT>** parameter. However, you can specify a range of states while using these options; the range will be interpreted and displayed correctly.

Defaults

If no parameters are given, the trace list is displayed starting with the first state that has not yet been displayed. The number of states displayed is identical to the number of states displayed by the last **tl** command.

For example, if the last trace list display was **tl -td 5**, then the next **tl** command will start the display at state 6 and display a total of five states.

The **-a** option is in effect by default, which causes the address field to display absolute addresses.

The trace list also defaults to the last disassembly state used (that is, if **-d** was specified previously in a **tl** command, it will continue).

Examples

Using the 68000 sample program from Appendix A, we will show some simple trace list examples.

First, we will set up a trigger at the start of the program and take a trace. Type:

```
M> tg addr=2000
M> t
```

```
Emulation trace started
```

```
M> r 2000
U> tl
```

The first **tl** command you issue after a trace has begun always displays the top of the trace:

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
   -1    000004    2000  supr data rd word                 ---   .
    0    002000    2479  supr prog                     0.400 uS   +
    1    002002    0000  supr prog                     0.400 uS   .
    2    002004    1000  supr prog                     0.400 uS   .
    3    002006    2679  supr prog                     0.400 uS   .
    4    001000    0000  supr data rd word             0.400 uS   .
    5    001002    3000  supr data rd word             0.400 uS   .
    6    002008    0000  supr prog                     0.400 uS   .
    7    00200a    1004  supr prog                     0.400 uS   .
    8    00200c    14bc  supr prog                     0.400 uS   .
```

```
U> tl
```

The next **tl** command issued starts with the first state not yet displayed:

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
    9    001004    0000  supr data rd word             0.400 uS   .
   10    001006    4000  supr data rd word             0.400 uS   .
   11    00200e    0000  supr prog                     0.400 uS   .
   12    002010    1012  supr prog                     0.400 uS   .
   13    003000      00  supr data wr byte             0.400 uS   .
   14    002012    0c00  supr prog                     0.400 uS   .
   15    003000      00  supr data rd byte             0.400 uS   .
   16    002014    0000  supr prog                     0.400 uS   .
   17    002016    67f8  supr prog                     0.400 uS   .
   18    002018    0c00  supr prog                     0.400 uS   .
```

If you now want to return to the top of the trace list and disassemble instructions, type:

```
U> tl -td
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
   -1    000004    2000  supr data rd word                 ---   .
    0    002000  MOVEA.L 0001000,A2                   0.400 uS   +
    1    002002    0000  supr prog                     0.400 uS   .
    2    002004    1000  supr prog                     0.400 uS   .
    3    002006  MOVEA.L 0001004,A3                   0.400 uS   .
    4    001000    0000  supr data rd word             0.400 uS   .
    5    001002    3000  supr data rd word             0.400 uS   .
    6    002008    0000  supr prog                     0.400 uS   .
    7    00200a    1004  supr prog                     0.400 uS   .
    8    00200c  MOVE.B  #000,[A2]                     0.400 uS   .
```

You can also vary the number of states displayed. Type:

```
U> tl -td 5
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
   -1    000004    2000  supr data rd word                 ---   .
    0    002000  MOVEA.L 0001000,A2                   0.400 uS   +
    1    002002    0000  supr prog                     0.400 uS   .
    2    002004    1000  supr prog                     0.400 uS   .
    3    002006  MOVEA.L 0001004,A3                   0.400 uS   .
```

Remember that **tl** always displays the same number of states displayed the last time **tl** was executed (but did not display the whole trace).

```
U> tl -n
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
    4    001000    0000  supr data rd word             0.400 uS   .
    5    001002    3000  supr data rd word             0.400 uS   .
    6    002008    0000  supr prog                     0.400 uS   .
    7    00200a    1004  supr prog                     0.400 uS   .
    8    00200c  MOVE.B  #000,[A2]                     0.400 uS   .
```

Notice that only five states were displayed.

You can also display a range of states:

```
U> tl -td 20..30
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
   20    002012  CMPI.B  #000,D0                      0.400 uS   .
   21    003000       00  supr data rd byte           0.400 uS   .
   22    002014     0000  supr prog                   0.400 uS   .
   23    002016  BEQ.B   0002010                      0.400 uS   .
   24    002018  CMPI.B  #**,D0                       0.400 uS   .
   25    002010  MOVE.B  [A2],D0                      0.600 uS   .
   26    002012  CMPI.B  #000,D0                      0.400 uS   .
   27    003000       00  supr data rd byte           0.400 uS   .
   28    002014     0000  supr prog                   0.400 uS   .
   29    002016  BEQ.B   0002010                      0.400 uS   .
   30    002018  CMPI.B  #**,D0                       0.400 uS   .
```

Remember, **tl** displays the same number of states displayed last time a partial trace was displayed, and starts with the next undisplayed state:

```
U> tl
```

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  ---------------------------------  ---------  ---
   31    002010  MOVE.B  [A2],D0                      0.600 uS   .
   32    002012  CMPI.B  #000,D0                      0.400 uS   .
   33    003000       00  supr data rd byte           0.400 uS   .
   34    002014     0000  supr prog                   0.400 uS   .
   35    002016  BEQ.B   0002010                      0.400 uS   .
   36    002018  CMPI.B  #**,D0                       0.400 uS   .
   37    002010  MOVE.B  [A2],D0                      0.600 uS   .
   38    002012  CMPI.B  #000,D0                      0.400 uS   .
   39    003000       00  supr data rd byte           0.400 uS   .
   40    002014     0000  supr prog                   0.400 uS   .
   41    002016  BEQ.B   0002010                      0.400 uS   .
```

Notice that 11 states were displayed, starting with state 31.

To suppress display of the column headers, use the **-h** option:

```
U> tl -h
```

```
42   002018  CMPI.B  #**,D0                    0.400 uS   .
43   002010  MOVE.B  [A2],D0                   0.600 uS   .
44   002012  CMPI.B  #000,D0                   0.400 uS   .
45   003000      00  supr data rd byte         0.400 uS   .
46   002014    0000  supr prog                 0.400 uS   .
47   002016  BEQ.B   0002010                   0.400 uS   .
48   002018  CMPI.B  #**,D0                    0.400 uS   .
49   002010  MOVE.B  [A2],D0                   0.600 uS   .
50   002012  CMPI.B  #000,D0                   0.400 uS   .
51   003000      00  supr data rd byte         0.400 uS   .
52   002014    0000  supr prog                 0.400 uS   .
```

You can also combine options (subject to the restrictions listed above under the note on binary and hexadecimal trace dump formats):

```
U> tl -hnd 12
```

```
53   002016  BEQ.B   0002010                   0.400 uS   .
54   002018  CMPI.B  #**,D0                    0.400 uS   .
55   002010  MOVE.B  [A2],D0                   0.600 uS   .
56   002012  CMPI.B  #000,D0                   0.400 uS   .
57   003000      00  supr data rd byte         0.400 uS   .
58   002014    0000  supr prog                 0.400 uS   .
59   002016  BEQ.B   0002010                   0.400 uS   .
60   002018  CMPI.B  #**,D0                    0.400 uS   .
61   002010  MOVE.B  [A2],D0                   0.600 uS   .
62   002012  CMPI.B  #000,D0                   0.400 uS   .
63   003000      00  supr data rd byte         0.400 uS   .
64   002014    0000  supr prog                 0.400 uS   .
```

Related Commands

t (starts an analyzer trace)

tf (specifies the display format for the trace)

th (halts a trace in process)

tlb (defines analyzer signal line labels; these may be used by **tf** in specifying the trace list display format)

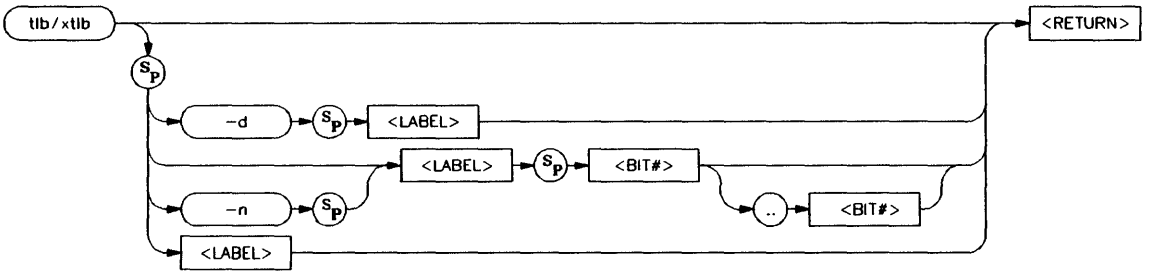
ts (allows you to determine the current status of the emulation analyzer)

tlb,xtlb

Summary

Define labels for analyzer input lines

Syntax



Function

The **tlb** (**xtlb**) command allows you to define new labels for emulation (external) analyzer lines, as well as display or delete previously defined analyzer labels. Since labels are pre-defined for the address, data, and status lines of the emulation analyzer, **xtlb** will be the more frequently used command.

<BIT>..<BIT> specifies the range of analyzer lines to be associated with <LABEL>. Note that it is not necessary to specify an upper boundary; if only one bit number is given, it is the only one that will be associated with the given label.

The external analyzer has 16 lines that may be assigned to labels, numbered 0 through 15, where 0 is the least significant bit. The emulation analyzer, dependent on the particular emulator in use, has between 32 and 80 lines, where 0 is the least significant bit.

In emulation analyzer labels, no more than 32 signal lines may be assigned to a given label. Also, an emulation analyzer label may not cross more than a multiple of 16 boundary. For example, a label cannot be defined for emulation analyzer lines 15..32 since

one multiple of 16 boundary is crossed from 15 to 16 and another boundary is crossed from 31 to 32.

Labels can be made to overlap; for example, you may wish to define a label for a particular status line or data bit so that you can easily track its state in the trace list. See examples below.

The number of labels that can be defined is limited only by system memory.

Parameters

-d	If you specify the **-d** option with a <LABEL>, the named label is deleted from the definition table. If the given <LABEL> is currently used in a trace specification or in the trace display format (tf command), it will not be deleted until removed from all of the specifications. If <LABEL> is given as *, all labels are deleted.
-n	Specifying **-n** causes the named <LABEL> to be defined with negative polarity. That is, after label definition, bits that are a one (1) refer to a signal lower than the threshold voltage and bits that are a zero (0) refer to a signal higher than the threshold voltage. If **-n** is not specified, the named <LABEL> defaults to positive polarity.
<LABEL>	You use <LABEL> to specify a name for the group of signals indicated by <BIT_RANGE>. <LABEL> is an alphanumeric designator; upper and lower case are distinguished. Labels can have up to 31 characters. If <LABEL> is supplied without an option, the named label is displayed; if <LABEL> is given as *, all of the label definitions are displayed.

<BIT#> <BIT#> specifies first the lower (or only), then upper, bits of the range to be assigned to the named <LABEL>. If more than one bit is specified (creating a range), the bit numbers are separated by two periods (..).

Defaults

If no parameters are specified, the current label definitions are displayed. Upon emulator powerup, or after a **tinit** command, the only label definitions are the address, data, and status labels needed to operate the emulation and optional external analyzer. All new label definitions default to positive polarity unless the **-n** option is given.

Examples

For the 68000 example program in appendix A, you can define a trace label overlapping the lower byte of the data label, then use this new label in a trace format specification so that you can see the message being written to the program's output area after a command is entered.

First, set the analyzer to complex configuration:

```
M> tcf -c
```

Now, you can define a label which will overlap the lower data bus byte. Type:

```
M> tlb lowerdata 40..47
```

To view the label definitions, type:

```
M> tlb
```

You will see:

```
#### Emulation trace labels
tlb addr 0..23
tlb data 32..47
tlb lowerdata 40..47
tlb stat 24..31
```

If you want to view only the output write data on the lower data byte in ASCII format, type:

```
M> tf lowerdata,A
```

To set up an analyzer specification, you must first define the pattern and ranges to trigger and qualify the trace. Type:

```
M> tpat p1 addr=4000
M> tpat p5 lowerdata!=0
M> trng addr=4000..4011
M> tg p1
```

The analyzer will trigger when it encounters address 4000 hex (the output area); it will only store data not equal to zero that is read from or written to the output area (thus, it will not store the CLEAR_LOOP activity, but only the LOOP activity).

Now start the trace:

```
M> t
```

```
Emulation trace started
```

Run the program and input a command to the program's input area by typing:

```
M> r 2000
U> m 3000=41
```

To view the results, type:

```
U> tl 0..17
```

```
 Line  lowerdata,A
-----  -----------
    0  .
    1  T
    2  H
    3  I
    4  S
    5  .
    6  I
    7  S
    8  .
    9  M
   10  E
   11  S
   12  S
   13  A
   14  G
   15  E
   16  .
   17  A
```

Related Commands

tf (used to specify the trace list format; **tlb** <LABEL> definitions can be specified as output columns in the trace listing through the **tf** command)

tpat (trace pattern definition; labels defined in **tlb** can be used in pattern definitions)

trng (trace range, used to specify a range of valid values to be used in a trace specification; labels defined by **tlb** may be used in defining the trace range)

xtv (threshold voltage setting for analyzer lines; **tlb** can be used to define positive and negative logic for labels encompassing those lines)

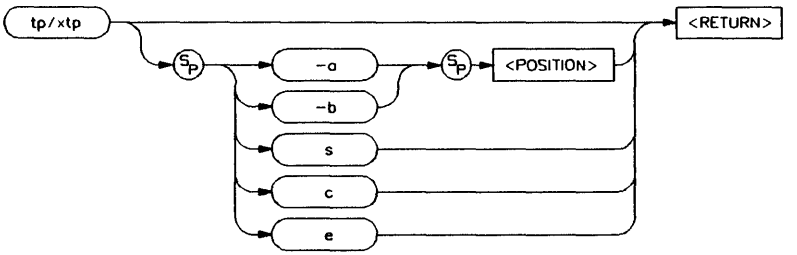
Notes

tp,xtp

Summary

Specifies location of trigger state in trace list

Syntax



Function

The **tp** (**xtp**) command allows you to specify where the trigger state will be positioned within the emulation (external) trace list.

If the trace tag counter (**tcq**) is disabled, the position number specified has an accuracy of +/- 3 states; otherwise, the accuracy is +/- 1 state.

Parameters

-a

Specifying **-a** along with a <POSITION> parameter indicates that the trigger is to be placed in the trace list with <POSITION> number of states after the trigger position to the end of the trace. That is, there will be <POSITION> number of states between the trigger position and the end of the trace. This option is invalid for the external analyzer set to timing mode (**xtmo -t**).

-b

Specifying **-b** along with a <POSITION> parameter indicates that the trigger is to be placed in the trace list with <POSITION>

number of states before the trigger position to the beginning of the trace. That is, there will be <POSITION> number of states between the beginning of the trace and the trigger position. This option is invalid for the optional external analyzer set to timing mode (**xtmo -t**).

<POSITION> — **<POSITION>** is a decimal value from 0 to 1023 (or 0 to 511 if **tcq** is in effect) specifying the number of states positioned before or after the trigger state, depending on the option supplied.

s — If you specify the **s** parameter, the trigger is positioned at the start of the trace list.

c — If you specify the **c** parameter, the trigger is positioned at the center of the trace list.

e — If you specify the **e** parameter, the trigger is positioned at the end of the trace list.

Note

The **s**, **c**, and **e** options are the only position parameters that are valid for the optional external analyzer set to timing mode (**xtmo -t**).

Defaults

If no parameters are supplied, the current trigger position setting is displayed. Upon powerup or after **tinit**, the trigger position is **tp s**.

Examples

The following examples were constructed using the 68000 sample program from Appendix A.

To display the current setting of the trigger position, type:

```
M> tp
```

You will see:

```
tp s
```

Now let's define a trigger and try various positioning methods to see the results. Type:

```
M> tg addr=2018
M> t
```

```
Emulation trace started
```

```
M> r 2000
M> m 3000=41
M> tl -d
```

```
  Line    addr,H   68000 Mnemonic                          count,R   seq
 -----    ------  ----------------------------------  ---------  ---
     0    002018  CMPI.B  #**,D0                            ---    +
     1    002010  MOVE.B  [A2],D0                       0.600 uS    .
     2    002012  CMPI.B  #000,D0                       0.400 uS    .
     3    003000       00  supr data rd byte            0.400 uS    .
     4    002014     0000  supr prog                    0.400 uS    .
     5    002016  BEQ.B   0002010                       0.400 uS    .
     6    002018  CMPI.B  #**,D0                       0.400 uS    .
     7    002010  MOVE.B  [A2],D0                       0.600 uS    .
     8    002012  CMPI.B  #000,D0                       0.400 uS    .
     9    003000       00  supr data rd byte            0.400 uS    .
```

Note that the trigger (always state zero (0)) is positioned at the start of the trace. Let's move it to the end of the trace and redo the trace by typing:

```
M> tp e
M> t
```

```
Emulation trace started
```

```
M> m 3000=41
M> tl -d -10..1
```

```
   Line    addr,H   68000 Mnemonic                          count,R   seq
  -----    ------  ----------------------------------  ---------  ---
     -4
     -3    003000       00  supr data rd byte                ---    .
     -2    002014  ORI.B   #0f8,D0                       0.400 uS    .
     -1    002016     67f8  supr prog                    0.400 uS    .
      0    002018  CMPI.B  #**,D0                       0.400 uS    +
      1
```

Here, the trigger has been positioned at the last state of the trace (note that state 1 is empty). Now position the trigger at the center of the trace list:

```
M> tp c
M> t
```

```
Emulation trace started
```

```
M> m 3000=41
M> tl -d -10..10
```

```
  Line    addr,H  68000 Mnemonic                          count,R  seq
 -----    ------  ---------------------------------  ---------  ---
    -3
    -2    002014    0000  supr prog                         ---   .
    -1    002016    67f8  supr prog                    0.400 uS   .
     0    002018    0c00  supr prog                    0.400 uS   +
     1    002010    1012  supr prog                    0.600 uS   .
     2    002012    0c00  supr prog                    0.400 uS   .
     3    003000      00  supr data rd byte            0.400 uS   .
     4    002014    0000  supr prog                    0.400 uS   .
     5    002016    67f8  supr prog                    0.400 uS   .
     6    002018    0c00  supr prog                    0.400 uS   .
     7    002010    1012  supr prog                    0.600 uS   .
     8    002012    0c00  supr prog                    0.400 uS   .
     9    003000      00  supr data rd byte            0.400 uS   .
    10    002014    0000  supr prog                    0.400 uS   .
```

Here the trigger is positioned approximately at the center of the trace list, with roughly 255 states preceding the trigger and 255 states following the trigger. (Note that not all of the states preceding the trigger have been filled.) Next, let's position the trigger using the "after" and "before" parameters. Type:

```
M> tp -a 10
M> t
```

```
Emulation trace started
```

```
M> m 3000=41
M> tl -d
```

```
 Line   addr,H  68000 Mnemonic                        count,R  seq
-----   ------  ----------------------------------  ---------  ---
    0   002018  CMPI.B  #**,D0                            ---   +
    1   002010  MOVE.B  [A2],D0                      0.600 uS   .
    2   002012  CMPI.B  #000,D0                      0.400 uS   .
    3   003000       00  supr data rd byte           0.400 uS   .
    4   002014     0000  supr prog                   0.400 uS   .
    5   002016  BEQ.B   0002010                      0.400 uS   .
    6   002018  CMPI.B  #**,D0                      0.400 uS   .
    7   002010  MOVE.B  [A2],D0                      0.600 uS   .
    8   002012  CMPI.B  #000,D0                      0.400 uS   .
    9   003000       00  supr data rd byte           0.400 uS   .
   10   002014     0000  supr prog                   0.400 uS   .
   11   002016  BEQ.B   0002010                      0.400 uS   .
   12
```

We asked for 10 states after the trigger to the end of trace and got 11. This is within the specified accuracy of the system. Now try the "before" parameter. Type:

```
M> tp -b 5
M> t
```

```
Emulation trace started
```

```
M> m 3000=41
M> tl -d
```

```
 Line   addr,H  68000 Mnemonic                        count,R  seq
-----   ------  ----------------------------------  ---------  ---
   -4   002012  CMPI.B  #000,D0                          ---   .
   -3   003000       00  supr data rd byte           0.400 uS   .
   -2   002014     0000  supr prog                   0.400 uS   .
   -1   002016  BEQ.B   0002010                      0.400 uS   .
    0   002018  CMPI.B  #**,D0                      0.400 uS   +
    1   002010  MOVE.B  [A2],D0                      0.600 uS   .
    2   002012  CMPI.B  #000,D0                      0.400 uS   .
    3   003000       00  supr data rd byte           0.400 uS   .
    4   002014     0000  supr prog                   0.400 uS   .
    5   002016  BEQ.B   0002010                      0.400 uS   .
    6   002018  CMPI.B  #**,D0                      0.400 uS   .
    7   002010  MOVE.B  [A2],D0                      0.600 uS   .
    8   002012  CMPI.B  #000,D0                      0.400 uS   .
    9   003000       00  supr data rd byte           0.400 uS   .
   10   002014     0000  supr prog                   0.400 uS   .
   11   002016  BEQ.B   0002010                      0.400 uS   .
   12   002018  CMPI.B  #**,D0                      0.400 uS   .
   13   002010  MOVE.B  [A2],D0                      0.600 uS   .
   14   002012  CMPI.B  #000,D0                      0.400 uS   .
   15   003000       00  supr data rd byte           0.400 uS   .
   16   002014     0000  supr prog                   0.400 uS   .
```

Here we specified five states before the trigger and got 4, which again is within the system's positioning accuracy.

Related Commands

tcq (used to specify the trace count qualifier; affects the number of states that can be stored by the analyzer)

tg (defines the trigger expression)

tl (used to display the trace list)

tsq (used to specify the trigger position within the trace sequencer; reference the sequencer operation when deciding where to position the trigger in the trace list, if you want to capture all of the sequence conditions)

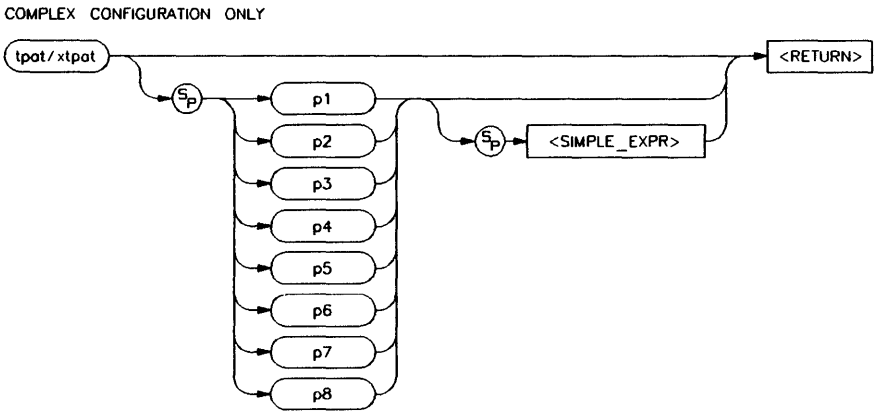
xtmo (specifies whether the external analyzer acts independently or is appended to the emulation analyzer)

tpat,xtpat

Summary

Specify analyzer complex configuration patterns

Syntax



Function

The **tpat** (**xtpat**) command allows you to assign pattern names to simple emulation (external) analyzer expressions. These pattern names are then used in building complex expressions for other analyzer commands.

The **tpat** command is only valid in the complex analyzer configuration (**tcf -c**).

Parameters

p1 - p8	The labels **p1** through **p8** are the names assigned to each simple expression. (The **p** in the label must be lowercase.)

<SIMPLE_EXPR> **<SIMPLE_EXPR>** lets you directly specify an analyzer expression to use as a storage qualifier. For example, **<SIMPLE_EXPR>** might consist of the expression **addr=2000**. For detailed information on specification of simple expressions, refer to the expression syntax pages.

Note 

Simple expressions assigned to patterns are restricted from the standard <SIMPLE_EXPR> definition in that you may not assign a range of values to a given label; only one value is permitted. (However, in actual practice, it is sometimes possible to circumvent this restriction by careful choice of don't care values in the expression.)

Also, patterns can be specified that encompass more bits than the number of bits defined for the specified label. When this occurs, the upper bits are truncated.

Defaults

If no parameters are given, or if the pattern name is given as *, all eight of the current pattern assignments are displayed. If one of the pattern names is given, the expression assigned to that pattern is displayed.

Upon entering complex configuration after powerup or a **tinit** initialization, all eight patterns are defined as **tpat <pattern#> any**.

Examples

If you're debugging the 68000 assembler program shown in Appendix A, you might wish to trigger the analyzer upon entering any of the output message setup routines, COMMAND_A, COMMAND_B, or UNRECOGNIZED. In addition, you can equate these procedure names to the address locations so they will be easier to remember.

First, set up the equates by typing:

```
M> equ commanda=202c
M> equ commandb=203a
M> equ unrecognized=2048
```

To use the pattern assignment command (**tpat**) you must put the analyzer in complex configuration. Type:

```
M> tcf -c
```

Now set up the pattern assignments using the previously defined equates:

```
M> tpat p1 addr=commanda
M> tpat p2 addr=commandb
M> tpat p3 addr=unrecognized
```

To set up a trigger when any one of the above patterns will trigger the analyzer, type:

```
M> tg p1|p2|p3
```

Here, the intraset OR operator (|) is used to relate the patterns. Refer to the syntax pages for <COMPLEX_EXPR> for details on how patterns are combined to create complex expressions.

In another instance (still using the 68000 program from Appendix A), you might want to be able to trigger the analyzer on various commands received at the command input location (3000 hex). First, set up the equates by typing:

```
M> equ inputpointer=3000
M> equ inputa=41
M> equ inputb=42
M> equ notacommand=00
```

Now set up various pattern combinations:

```
M> tpat p1 addr=inputpointer and data=inputa
M> tpat p2 addr=inputpointer and data=inputb
M> tpat p3 addr=inputpointer
M> tpat p5 data=inputa
M> tpat p6 data=inputb
M> tpat p7 data!=notacommand
M> tpat p8 data=notacommand
```

To trigger the analyzer when address=3000 and data=41 (an "A" command):

```
M> tg p1
```

Or, you could trigger on a "B" command:

```
M> tg p2
```

If you want to trigger when an unrecognized command is read, type:

```
M> tg p3 and p5~p6~p8
```

Or, you could trigger when either command "A" or command "B" is read:

```
M> tg p3 and p5|p6
```

You might want to trigger if any command is read:

```
M> tg p3 and p7
```

You should note that it is NOT necessary to use equates to associate names with numeric patterns. To define **p1** and **p2** above, you could type the following with the same results:

```
M> tpat p1 addr=3000 and data=41
M> tpat p2 addr=3000 and data=42
```

Related Commands

tcf (defines whether the analyzer is in easy configuration or complex configuration; the **tpat** command is only valid in complex configuration)

tcq (specifies a trace count qualifier; **tpat** patterns may be used in complex configuration qualifier specification)

telif (specifies a secondary branch qualifier in analyzer complex configuration; **tpat** patterns may be used in qualifier specification)

tg (used to specify a simple trigger in either easy configuration or complex configuration; **tpat** patterns may be used in complex configuration trigger specification)

tif (used to specify a primary branch qualifier in either analyzer configuration; **tpat** patterns may be used in complex configuration branch specifications)

tpq (specifies a trace prestore qualifier; **tpat** patterns may be used in qualifier specification)

trng (defines a range of values on a set of analyzer input lines; this range may be used in conjunction with the patterns defined by **tpat** in setting up complex analysis qualifiers)

tsq (used to manipulate the trace sequencer)

tsto (used to define global storage qualifiers in both analyzer configurations; may also be used to define storage qualifiers for each sequencer level in complex configuration. The patterns defined by **tpat** may be used in complex configuration storage qualifier definition.)

xtmo (determines whether the external analyzer acts as an independent state or timing analyzer or is appended to the emulation analyzer. If appended, the **xtpat** command is no longer valid; **tpat** defines patterns to be used across both analyzers.)

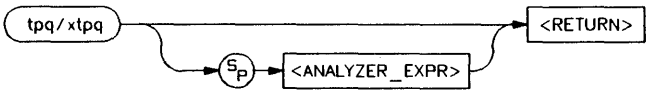
Notes

tpq,xtpq

Summary

Specify trace prestore qualifier

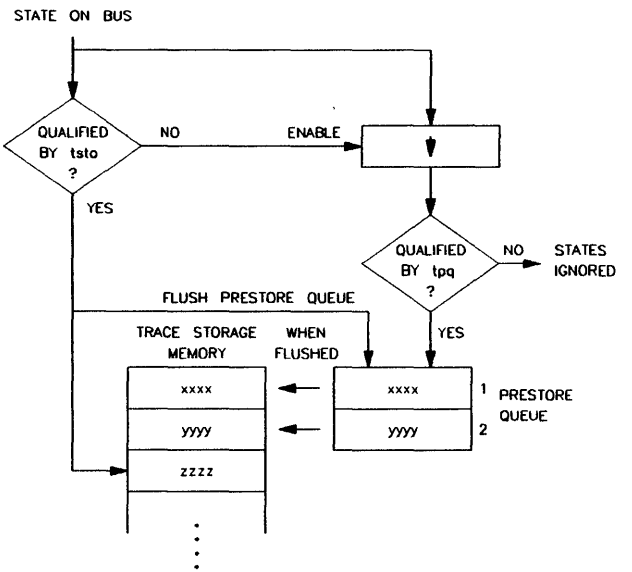
Syntax



Function

The **tpq** (**xtpq**) command allows you to specify a prestore qualifier for the emulation (external) trace.

During the trace, the analyzer fills a two stage pipe with states that satisfy the prestore qualifier. Each time a trace state is stored into the trace buffer, the prestore qualifier is also stored and then cleared. Therefore, up to two prestore events may be stored for



each normal store event; the prestore events in the trace buffer will correspond to the most recent states that satisfied the prestore qualifier immediately prior to a store event but following the previous store event.

Since the prestore memory shares trace memory with store events, the number of store events recorded will be reduced by the number of prestore states recorded.

Parameters

<ANALYZER_EXPR>

<ANALYZER_EXPR> allows you to specify the expression to be recognized as a prestore state. This expression consists of a **<SIMPLE_EXPR>** in analyzer easy configuration and a **<COMPLEX_EXPR>** when the analyzer is in complex configuration. Refer to the syntax pages for expressions for specific details of analyzer expressions. In either configuration, the expression may consist of the states **any** (prestore all states) or **none** (disable prestore).

Defaults

If no parameters are given, the current prestore qualifier setting is displayed. Upon powerup or after **tinit** initialization, the prestore qualifier defaults to **tpq none**.

Examples

With the example 68000 program given in Appendix A, you might want to prestore the command input that leads to the output routines. To do this, you first need to set up a trigger state:

```
U> tg addr=2000
```

Then, set up a storage qualifier that will store only the message setup and output routines:

```
U> tsto addr=202c..2071
```

Note

Specifying a storage qualifier of **tsto any** will not produce the desired results; the prestore qualifier will be ignored in favor of the regular storage event.

Now set up the prestore qualifier by typing:

```
U> tpq addr=3000
```

Now you can proceed with the measurement. Type:

```
U> t
```

```
Emulation trace started
```

```
U> r 2000
U> m 3000=21
```

(This inputs a "command" value to the program.)

```
U> tl -d
```

```
  Line    addr,H  68000 Mnemonic                          count,R  seq
 -----    ------  ---------------------------------  ---------  ---
     0    002000  MOVEA.L ********,A2                       ---   +
     1    003000       00  supr data rd byte            prestore   .
     2    003000       21  supr data rd byte            prestore   .
     3    002048  MOVE.B  #00f,D0                       4.800  S   .
     4    00204a    000f  supr prog                     0.400 uS   .
     5    00204c  MOVEA.L #00000102a,A0                 0.400 uS   .
     6    00204e    0000  supr prog                     0.400 uS   .
     7    002050    102a  supr prog                     0.400 uS   .
     8    002052  MOVEA.L A3,A1                         0.400 uS   .
     9    002054  MOVE.B  #020,D1                       0.400 uS   .
```

The prestore events are shown on lines 1 and 2.

You can make similar measurements within the analyzer's complex configuration. Using the pattern specification capability, you can narrow down the prestore selections. For example, you might want to prestore only those command inputs which are "unrecognized"; that is, they are neither 41 hex, 42 hex, or 00 hex. First, you must set the analyzer to complex configuration. Type:

```
U> tcf -c
```

When the analyzer enters complex configuration, the **tcq** count qualifier is set to **none**. If it is not initialized to some other qualifier, the prestore labels will not be shown in the trace display. For this example, you can use the count time qualifier. Type:

```
U> tcq time
```

Now you need to define patterns for use in the trigger, store, and prestore qualifiers. Type:

```
U> tpat p1 addr=2000
U> tpat p2 addr=3000
U> trng addr=202c..2071
U> tlb lowerdata 40..47
U> tpat p5 lowerdata=41
U> tpat p6 lowerdata=42
U> tpat p7 lowerdata=00
```

To set up the prestore qualifier so that only "unrecognized" commands will be prestored, you need to construct an expression where address =3000 and data is not equal to 41 or 42 or 00. Type:

```
U> tpq p2 and p5~p6~p7
```

Next, set up the analyzer trigger condition:

```
U> tif 1 p1 2
U> tif 2 never
U> tsq -t 2
```

You want the analyzer to store only the output routines. (If you let the analyzer store all information, the prestore information will be overridden in favor of the regular store states.) Type:

```
U> tsto 2 r
```

To proceed with the measurement, type:

```
U> t
```

```
Emulation trace started
```

```
U> r 2000
U> m 3000=41
```

Here, we have input a "command" of 41 -- one of the values that should NOT result in a prestore state.

```
U> tl -d
```

```
 Line   addr,H  68000 Mnemonic                          count,R  seq
-----   ------  ---------------------------------  ---------  ---
    0   002000  MOVEA.L ********,A2                        ---   +
    1   00202c  MOVE.B  #011,D0                       3.054  S   .
    2   00202e    0011  supr prog                     0.400 uS   .
    3   002030  MOVEA.L #000001008,A0                 0.400 uS   .
    4   002032    0000  supr prog                     0.400 uS   .
    5   002034    1008  supr prog                     0.400 uS   .
    6   002036  BRA.W   0002052                       0.400 uS   .
    7   002038    001a  supr prog                     0.400 uS   .
    8   002052  MOVEA.L A3,A1                         0.600 uS   .
    9   002054  MOVE.B  #020,D1                       0.400 uS   .
```

Since the command input (41 hex) did not satisfy the prestore qualifier, no prestore states are shown in the trace listing. Now you can try it again with an "unrecognized" command. Type:

```
U> t
Emulation trace started

U> r 2000
U> m 3000=21
U> tl -d
```

```
 Line   addr,H  68000 Mnemonic                          count,R  seq
-----   ------  ---------------------------------  ---------  ---
   -1   000004    2000  supr data rd word                  ---   .
    0   002000  MOVEA.L ********,A2                  0.400 uS   +
    1   003000      21  supr data rd byte            prestore   .
    2   002048  MOVE.B  #00f,D0                       3.527  S   .
    3   00204a    000f  supr prog                     0.400 uS   .
    4   00204c  MOVEA.L #00000102a,A0                 0.400 uS   .
    5   00204e    0000  supr prog                     0.400 uS   .
    6   002050    102a  supr prog                     0.400 uS   .
    7   002052  MOVEA.L A3,A1                         0.400 uS   .
    8   002054  MOVE.B  #020,D1                       0.400 uS   .
```

The prestore state is shown on line 1 of the trace listing.

Related Commands

tcf (specifies whether the analyzer is to operate in easy configuration or complex configuration)

tsq (used to manipulate the trace sequencer)

tsto (used to specify a global storage qualifier for both easy configuration and complex configuration; also used to specify individual sequence term storage qualifiers in complex configuration)

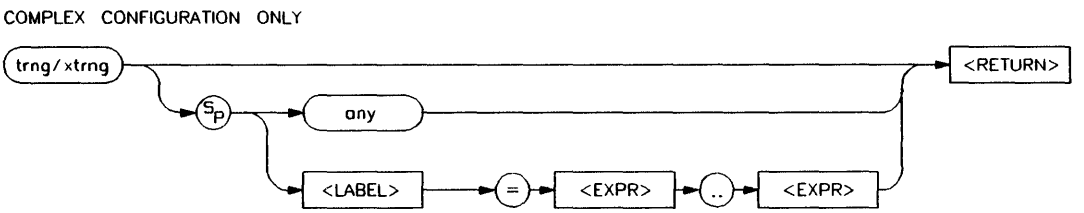
xtmo (specifies whether the external analyzer will act as an independent state or timing analyzer or whether it will be appended to the emulation analyzer. If appended to the emulation analyzer, the **xtpq** command has no effect; the **tpq** command sets the prestore qualifier for both analyzers.)

trng,xtrng

Summary

Specify a complex configuration range qualifier

Syntax



Function

The **trng** (**xtrng**) command lets you specify a range of acceptable values for an emulation (external) trace label. This range may then be used in complex qualifiers for the trace specification. The **trng** (**xtrng**) command is only available in the analyzer's complex configuration (see **tcf** syntax pages).

There is no need for a not equals operator in specifying ranges, as the trace specification commands which allow "range" as a parameter also accept "not range" in the form **!r**.

If the optional external analyzer has been appended to the emulation analyzer via the **xtmo** command, the **xtrng** command is invalid; **trng** sets a range pattern to be used by both analyzers.

Parameters

any	When you specify **any**, all possible patterns on all labels will satisfy the range specification.
<LABEL>	**<LABEL>** specifies the group of signal lines to which a range is assigned. These might be **addr, data,** or **stat**; or, they may be a label that you have defined. See the **tlb** command

syntax pages for information on defining labels.

<EXPR>

<EXPR> allows you to specify first the lower, then upper, boundaries of the range of patterns to be considered valid range entries. For example, to define the address range of 2000 through 21ff hex, you would specify the **<EXPR>** range as **2000..21ff**. Note the two periods used as a separator between the lower and upper range bounds; no additional spaces are included.

Also, the first boundary specified must be less than or equal to the second boundary specified (example: **trng addr=2000..21ff** is correct; **trng addr=21ff..2000** is incorrect). You may also specify a single value for the range (example: **trng addr=2000**).

Refer to the <EXPR> syntax pages in this manual for details on expression syntax.

Ranges can be specified that encompass more bits than the number of bits defined for the specified label.

Defaults

If no parameters are supplied, the current range definition is displayed. After powerup or **tinit** initialization, the **trng** command is set to **trng any**. (Note that **trng** is not directly available after analyzer initialization; the analyzer is set to easy configuration when initialized. You must then switch to complex configuration to access **trng**.)

Note

The **tcf -e** (set trace configuration to easy) command also will reset **trng**. In other words, any **trng** defined when the analyzer was in complex configuration is destroyed when the analyzer is set to easy configuration; you cannot return to complex configuration and use the old **trng**.

Examples

With the 68000 sample program in Appendix A, you may want to trigger the analyzer on any access to the message storage area located from 1008 through 1038 hexadecimal. To do this, type the following commands:

```
M> tcf -c
M> trng addr=1008..1038
M> tg r
M> t
```

```
Emulation trace started
```

```
M> r 2000
U> m 3000=41
U> tl -d
```

Line	addr,H	68000 Mnemonic		count,R	seq
0	001008	54	supr data rd byte	---	+
1	004000	54	supr data wr byte	0.400 uS	.
2	002068	ORI.B	#0f8,D1	0.400 uS	.
3	00206a	66f8	supr prog	0.400 uS	.
4	00206c	JMP	********	0.400 uS	.
5	002064	MOVE.B	[A0]+,[A1]+	0.600 uS	.
6	002066	SUBI.W	#00001,D0	0.400 uS	.
7	001009	48	supr data rd byte	0.400 uS	.
8	004001	48	supr data wr byte	0.400 uS	.
9	002068	0001	supr prog	0.400 uS	.

The analyzer is set to complex configuration (it must be in complex configuration to use the **trng** command); then "range" is defined as the addresses from 1008 through 1038 hex. The emulation analyzer trigger is then defined as the occurrence of range; a trace is started. You can see that the trigger was found in line number 0 of the trace listing; the program read the first byte of the message from the data area.

Related Commands

tcf (sets analyzer to complex or easy configuration; analyzer must be in complex configuration to utilize the **trng** command)

tcq (trace state/time counter; in complex configuration, states can be counted using the range specification)

telif (specifies the sequencer secondary branch expression; in complex configuration, this expression can include references to the range)

tg (specifies analyzer trigger; may trigger on references to range)

tif (specifies the sequencer primary branch expression; in complex configuration, branch expression may include range qualifier)

tpat (trace pattern definition; assigns pattern names to simple expressions for later use in analyzer specifications. **tpat** essentially commits only one pattern to a label; whereas **trng** allows a range of values to be assigned to the range pattern)

tpq (defines trace prestore qualifier; the range specification may be used in complex configuration prestore qualifier expressions)

tsq (trace sequencer definition)

tsto (defines trace storage qualifier; that is, specifies exactly what states are actually to be stored by the analyzer. In complex configuration, this can include states that fall within the specification defined by **trng**)

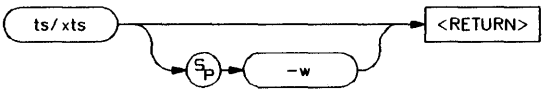
xtmo (specifies the mode of the external analyzer; either an independent state or timing analyzer or an analyzer appended to the emulation analyzer)

ts,xts

Summary

Display status of analysis trace

Syntax



Function

The **ts** (**xts**) command allows you to determine the current status of the emulation (external) analyzer.

Trace Status Displays

The emulation and external state trace status is displayed in the following form:

```
---[Emulation | External] Trace Status---
(NEW) [User | CMB ] trace [complete | halted | running ]
Arm [ ignored | (not) received ]
Trigger (not) found
Arm to trigger armcount
States visible (history) first..last
Sequence term term
Count remaining count
```

The external timing trace status is displayed in the following form:

```
--- External Timing Trace Status---
(NEW) [User | CMB ] trace [complete | halted | running ]
Arm [ ignored | (not) received ]
trace status
Arm to trigger armcount
Samples visible (history) first..last
```

The trace status header indicates whether this status is for the emulation or external state trace.

Whether the trace status is displayed as Emulation or External depends on:

- Presence of the optional external analyzer.
- Whether you entered the **ts** (emulation trace status) or **xts** (external trace status) command.
- The current mode setting of the optional external analyzer. If set as a state analyzer (**xtmo -s**), you can have an external state trace status. If set as a timing analyzer (**xtmo -t**), there is a different display for timing status (described below). If appended to the emulation analyzer, the **xts** command is invalid; the external analyzer acts as an extension to the emulation analyzer and their status is reported under the Emulation Trace Status.

Status Display Interpretation

The first line of the trace status indicates the initiator of the trace, whether the trace is completed, running, or halted, and whether or not this trace has been displayed.

NEW	This trace has not been displayed. The **tl** (**xtl**) command will clear this flag until the next trace is started. Halting a trace that is running (as opposed to complete), marks the trace as being NEW even though the trace may have been displayed while running. The next **tl** command with no options will list the trace from the top.
User	The operator initiated this trace with the **t** (**xt**) command.
CMB	This trace was initiated by a /EXECUTE pulse on the CMB after a **tx** command was entered.
complete	The trace has found its trigger and completed.

halted	The trace was halted in response to a **th** (**xth**) command.
running	The trace is still running; either the complete sequencer specifications have not yet been satisfied; or not enough qualified store states have been found to fill trace memory.

The second line of the trace display indicates the analyzer arm status.

ignored	The arm condition specified for this trace was **tarm always.**
received	The arm condition has been satisfied.
not received	The arm condition was not satisfied. (If you specified an arm condition but didn't use it in trigger qualification, this will be displayed if the arm condition is not satisfied. However, the analyzer may still find the correct trigger and complete the trace.)

The third line of the state trace display indicates the trigger status. Because of the pipelined analyzer architecture, it is possible that the trace status may display "not found" when in fact the trigger has been found. This will occur when not enough states satisfying the storage specification are found to push the trigger out of the pipeline and into trace memory. In any case, the trace will not be displayable until the trigger is in trace memory (unless you halt the analyzer).

found	The trigger condition has been found.
not found	The trigger condition has not yet been satisfied.

For the external timing status, the third line indicates the timing trace status. This will be one of the following strings:

Tracepoint found

Trigger found - delaying

Pattern found - waiting for edge

Prestore complete - waiting for trigger

Waiting for prestore

Waiting for arm

The fourth line of the trace display indicates the amount of time that passed between the arm signal and the trigger condition.

armcount This will be from -0.04 usec to 41.94288 ms. The arm to trigger counter may underflow or overflow, in which case "<-0.04 uS" or ">41.94288 mS" are reported, respectively. If the arm signal was ignored, if the trigger was not found, or if the clock setting (tck/xtck) is fast (F) or very fast (VF), the character "?" (unknown) is displayed.

The fifth line of the trace display indicates the number of states displayable by **tl**. (Number of samples in the case of the external timing trace.)

visible Number of states which can be displayed by tl (xtl); this will be a number from 0 to 1024 (or 0 to 512 if tcq is active).

history Number of states which can be displayed if the current trace is halted; this may include history states which may be overwritten and thus unavailable if the current trace runs to completion.

first Number of the first state stored in trace memory, relative to the trigger state. This will be a number from -1024 to 0 (-512 to 0 if

tcq is active). The character "?" is displayed if the trigger state is not yet in memory.

last Number of the last state stored in trace memory, relative to the trigger state. This will be a number from -1 to 1023 (-1 to 511 if **tcq** is active). The character **?** is displayed if the trigger state is not yet in memory.

The sixth line of the trace display indicates the current sequencer term position. (Not used in the external timing trace status.)

term Current sequence term position (1 through 5 in easy configuration; 1 through 8 in complex configuration). If the trace is completed or halted, the last sequence term number is displayed. A "?" is displayed if the trace is running and the sequencer is running too quickly for the current term number to be read.

The seventh line of the trace display indicates the count qualifier status for the primary branch condition of the current sequence term, see **tif** for further details. (Not used in the external timing trace status.)

count Remaining number of occurrences of the primary branch qualifier needed to satisfy the qualifier so that the primary branch will be taken. A "?" is displayed if the trace is running and the counter is updating too quickly to be read.

Whisper Mode Trace Display

If the **-w** option is given, an abbreviated version of the trace status is given as follows:

Trace run status:

R - trace running

C - trace completed

H - trace halted

Trace arm status:

A - Arm has been received

a - arm has not yet been received

x - arm signal is being ignored

Trace trigger status:

T - trace trigger has been found

t - trace trigger has not yet been found

Trace list status:

***** - indicates that this trace has not been displayed

Parameters

-w	The **-w** option indicates that the trace status should be printed in whisper mode; this gives an abbreviated version of the status. See "Function" above for interpretation of the whisper status information.

Defaults

If the whisper option is not specified, the long version of trace status is displayed.

Examples

Let's first start then halt a trace to look at the status. Type:

```
M> tg
```

```
tg any
```

```
M> r 2000
U> t
```

You will see:

```
Emulation trace started
```

Now type:

```
U> th
```

You will see:

```
Emulation trace halted
```

To view the trace status, type:

```
U> ts
```

You will see:

```
--- Emulation Trace Status ---
NEW User trace complete
Arm ignored
Trigger found
Arm to trigger ?
States 512 (512) 0..511
Sequence term 2
Occurrence left 1
```

Here, the trace was initiated by the user and has been completed but not yet displayed. The arm condition is disabled and the trigger was found. Since no arm was specified, the system cannot determine the amount of time between the arm and the trigger, so it is displayed as a question mark ?. There are 512 states in trace memory, with 0 being the first state with respect to the trigger and 511 being the last. (Note that since there are only 512 states, the trace count qualifier must be enabled). The sequencer is on the second sequence term (which is the trigger term for this example); the occurrence count is one (1) since the sequencer has found the trigger term (remember that the last primary branch qualifier (the trigger term) is **tif 5 never**).

Now, display the trace list by typing:

```
U> tl -d
```

You will see a display such as the following:

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  --------------------------------  ---------  ---
    0    003000       00  supr data rd byte              ---   +
    1    002014  ORI.B    #0f8,D0                    0.400 uS   .
    2    002016    67f8   supr prog                   0.400 uS   .
    3    002018  CMPI.B   #**,D0                      0.400 uS   .
    4    002010  MOVE.B   [A2],D0                     0.600 uS   .
    5    002012  CMPI.B   #000,D0                     0.400 uS   .
    6    003000       00  supr data rd byte           0.400 uS   .
    7    002014     0000  supr prog                   0.400 uS   .
    8    002016  BEQ.B    0002010                    0.400 uS   .
    9    002018  CMPI.B   #**,D0                      0.400 uS   .
```

You can also display the short form of the status above by typing:

```
U> ts -w
```

You will see:

```
CxT
```

Note that there is no * symbol. That is because we just issued a **tl** command to display the trace.

Let's look at another example. Set a trace trigger to a value that won't be found:

```
U> tg never
```

Now initiate a trace by typing:

```
U> t
```

```
Emulation trace started
```

To obtain the new trace status, type:

```
U> ts
```

You will see:

```
--- Emulation Trace Status ---
NEW User trace running
Arm ignored
Trigger not found
Arm to trigger ?
States ? (512) ?..?
Sequence term 1
Occurrence left 1
```

Now, we have a trace in process, initiated by the user; the trace list has not yet been displayed. The arm condition is disabled; the trigger has not yet been found. There are 512 states in trace memory; however, since there is no trigger condition, the states have not been numbered and will be overwritten by subsequently occurring states. The sequencer is still on term one, looking for one occurrence of the trigger pattern.

Display the short version of the same status by typing:

```
U> ts -w
```

You will see:

```
Rxt*
```

Note the * which indicates that the current trace list has not been displayed.

Now halt the trace in process by typing:

```
U> th
```

```
Emulation trace halted
```

You can display the trace list by typing:

```
U> tl -d
```

You will see:

```
 Line    addr,H  68000 Mnemonic                          count,R  seq
-----    ------  --------------------------------  ---------  ---
 -512    002014  ORI.B   #0f8,D0                          ---   .
 -511    002016    67f8  supr prog                    0.400 uS   .
 -510    002018  CMPI.B  #**,D0                       0.400 uS   .
 -509    002010  MOVE.B  [A2],D0                      0.600 uS   .
 -508    002012  CMPI.B  #000,D0                      0.400 uS   .
 -507    003000      00  supr data rd byte            0.400 uS   .
 -506    002014    0000  supr prog                    0.400 uS   .
 -505    002016  BEQ.B   0002010                      0.400 uS   .
 -504    002018  CMPI.B  #**,D0                       0.400 uS   .
 -503    002010  MOVE.B  [A2],D0                      0.600 uS   .
```

Now display the new trace status:

```
U> ts
```

```
--- Emulation Trace Status ---
User trace halted
Arm ignored
Trigger not found
Arm to trigger ?
States 512 (512) -512..-1
Sequence term 1
Occurrence left 1
```

Now the trace has been halted. The NEW flag is gone; therefore, the current trace list has been displayed. The trigger was never found. The "States" list indicates that 512 states are in memory; these are the last 512 states recorded by the analyzer up to the point where the analyzer was halted.

To display the short form of this status, type:

```
U> ts -w
```

You will see:

```
Hxt
```

For the external timing analyzer, a typical trace status display might be:

```
--- External Timing Trace Status ---
NEW User trace complete
Arm ignored
Tracepoint found
Arm to trigger ?
Samples 1024 (1024), -59..964
```

Related Commands

es (allows you to determine general emulator status)

t (starts an emulation trace)

tarm (arm the analyzer based on state of the trig1 and trig2 signals)

tcq (specify trace tag counter; affects number of states that the analyzer can store)

tg (specify the analyzer trigger state)

th (halt the current trace in process)

tif (specify sequencer primary branch condition and number of occurrences)

tx (specify that trace is to begin upon receiving the CMB /EXECUTE pulse)

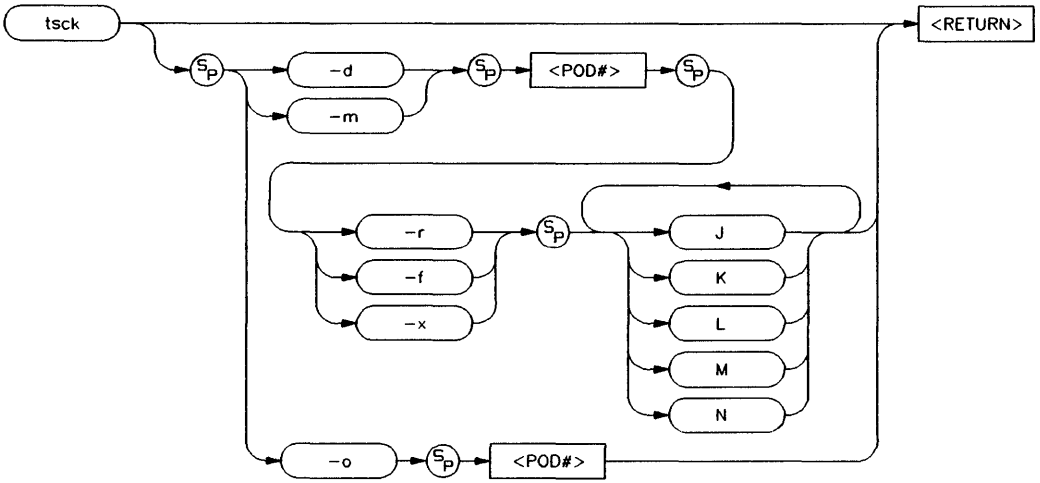
x (begin a synchronous CMB execution)

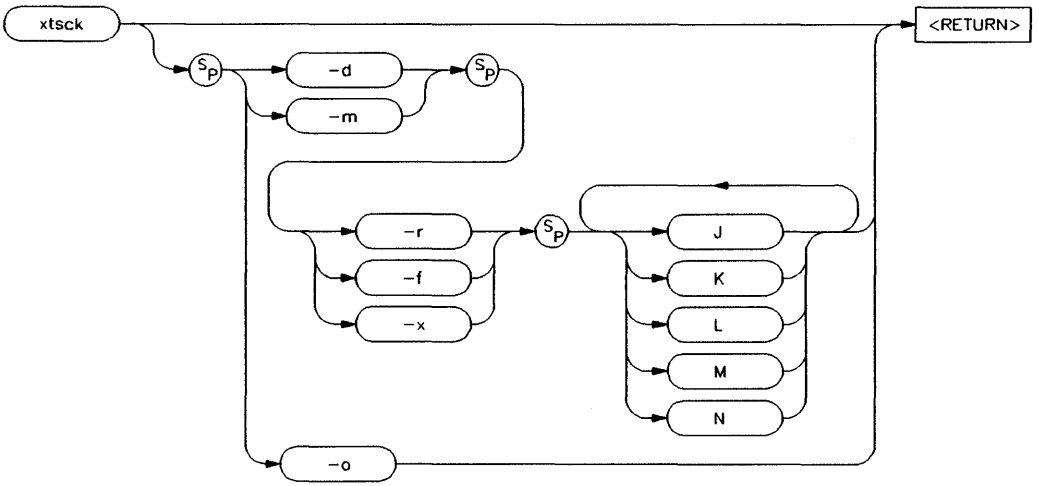
tsck,xtsck

Summary

Specify analyzer slave clocks

Syntax





Function

The **tsck** (**xtsck**) command allows you to specify the slave clock edges used for the emulation (external) analyzer trace.

Each analyzer pod has the capability of latching certain signals with a slave clock instead of the master clock. (You set up the master clock with the **tck** command.)

The **xtsck** command controls the slave clock for the optional external analyzer. No pod number is necessary since the external analyzer has only one pod.

Parameters

d

The **-d** option allows you to specify that the slave clock operates in demultiplexed mode. In this mode, the lower 8 channels of the analyzer pod (bits 0-7) are latched with the slave clock and the upper 8 channels (bits 8 through 15) are replaced with the lower 8 channels. In other words, the upper 8 bits are identical to the lower 8 at the pod.

However, the data is not clocked into the analyzer itself until the next master clock occurs. Therefore, if no slave clocks have occurred since the last master clock, the data on the lower 8 analyzer lines is identical to the upper 8. If one or more slave clocks have occurred since the last master clock, the data on the lower 8 bits is the only data available to the analyzer.

When using the **-d** option, you must specify one of the **-r**, **-f**, or **-x** options to indicate the active edge(s) of the slave clock.

m

The **-m** option specifies that the slave clock operates in mixed mode. In the mixed mode, the lower 8 channels of the analyzer pod (bits 0-7) are latched with the slave clock, and the master clock latches in the entire pod. Therefore, if no slave clock has

occurred since the last master clock, the data on the lower 8 bits of the pod will be clocked into the analyzer at the same time as the upper 8 bits. If more than one slave clocks has occurred since the last master clock, only the first slave clock data will be available to the analyzer.

When using the **-m** option, you must specify one of the **-r**, **-f**, or **-x** options to indicate the active edge(s) of the slave clock.

<POD#> Specifies one of 5 groups of analyzer input lines. These are as follows:

Pod #	Analyzer	Bits
1	Emulation	0 - 15
2	Emulation	16 - 31
3	Emulation	32 - 47
4	Emulation	48 - 63
5	External	0 - 15

Note that you only need to specify pod 5 if you are using the **tsck** command to operate on the optional external analyzer. You would typically do this only if you had logically joined the analyzers using the **xtmo** command.

r Indicates that the pod should latch data on the **rising** edge of the slave clock.

f Indicates that the pod should latch data on the **falling** edge of the slave clock.

x	Indicates that the pod should latch data on **both** edges of the slave clock.
CLOCK SIGNALS	The **r**, **f**, and **x** operators may be used on the following clock signals: **J, K, L, M** or **N**. Clocks **L**, **M**, and **N** are generated by the emulator. Clocks **J** and **K** are the external clock inputs on the optional external analyzer's probe. You should only use the external clock signals in clocking the external state/timing trace; they should not be used in clocking the emulation analyzer trace. You may use **L** and **M** to clock the external state trace as well as the emulator trace. If you specify multiple clocks, any one of the clock edges (as defined by the **r**, **f**, and **x** options) will clock the trace.
o	If you specify **-o** with a **<POD#>**, the slave clock is ignored on that pod. Remember that you don't need to specify **<POD#>** with the **xtsck** command; this command operates only on the single external analyzer pod.

Defaults

If no parameters are specified, the current slave clock definitions are displayed. The default for all slave clocks is **off** after powerup or **tinit** initialization.

Examples

To demultiplex pod 5 (external analyzer bits 0-15) with both edges of the **J** clock, type:

```
M> tsck -d 5 -x J
```

To display the current state of the slave clock specifications, type:

```
M> tsck
```

You will see:

```
tsck -o 1
tsck -o 2
tsck -o 3
tsck -o 4
tsck -d 5 -x J
```

Related Commands

ta (allows you to display active signals on the analyzer input lines; useful in verifying that you have selected the correct clock conditions)

tck (used to define master clock signals used by the analyzer; **tsck** defines the slave clock signals. Default mode for **tsck** is off on all pods.)

xtv (specifies threshold voltages for external analyzer input lines; must be set correctly to ensure that the **J** and **K** clock signals are recognized)

xtmo (specifies mode of operation for the external analyzer; that is, whether it acts as an independent analyzer or is appended to the emulation analyzer)

Notes

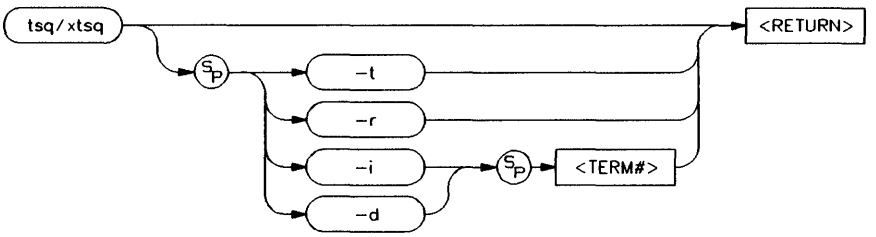
tsq,xtsq

Summary

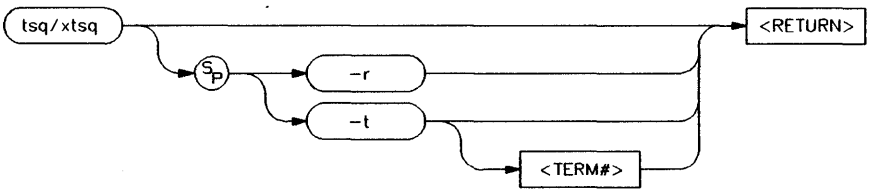
Manipulate the trace sequencer

Syntax

EASY CONFIGURATION



COMPLEX CONFIGURATION



Function

The **tsq** (**xtsq**) command allows you to manipulate or display the emulation (external) trace sequencer.

When the analyzer is in easy configuration (**tcf -e**), the sequencer has a maximum of four sequence terms with a minimum of one term.

If the analyzer is in complex configuration (**tcf -c**), the sequencer always has eight terms (although the particular sequencer setup may mean that only two are ever accessed).

Parameters (Easy Configuration)

t

If the **t** option is specified, the trigger term is displayed. The trace trigger occurs the first time that this term is entered during a trace. When in easy configuration, the trigger condition is always the primary branch condition for the last term in the defined sequence.

r

If you specify **r**, the sequencer is reset to a simple one term sequence which stores all states and triggers on the first occurrence of any state. This is equivalent to issuing the commands:

```
tg any
tsto any
telif never
```

i

Specifying **i** in conjunction with a **<TERM#>** inserts a new sequence term at **<TERM#>**. The new sequence term will use the default storage qualifier (which can be modified with the **tsto** command). It will also use the secondary branch qualifier (global restart in easy configuration) specified by the **telif** command.

If there is already a sequence term with number **<TERM#>**, terms with number **<TERM#>** and above will be renumbered (**<TERM#>** becomes **<TERM#> + 1**) to make room for the new term.

The primary branch qualifier for the new term will be defined as **tif <TERM#>** any unless it is the last term in the sequence (by definition, the trigger term), in which case

the primary branch qualifier is set to **tif <TERM#> never**.

d — Specifying **d** in conjunction with a **<TERM#>** deletes the term specified and renumbers higher numbered terms downward to fill the gap.

<TERM#> — **<TERM#>** specifies a term number in the range 1 through 4 to insert in the sequencer (**-i**) or remove from the sequencer (**-d**). You must insert terms in a contiguous manner; for example, you cannot insert a term number 4 if the sequencer only has two terms defined. Instead, you must next insert a term numbered 1, 2 or 3.

Parameters (Complex Configuration)

r — If you specify **-r**, the sequencer is reset to an eight term sequence with the trigger term at term number 2. The sequencer will be set to **tsto any** (store any state). All secondary branch qualifiers are turned off (**telif <TERM#> never**), and all primary branch qualifiers will jump to the next higher numbered term on any state (**tif <TERM#> any (<TERM#> +1)**).

t — Specifying **-t** by itself displays the trigger term. You can define which term is to be the trigger term by specifying **-t** along with a **<TERM#>**. The analyzer will trigger on the first entrance to the term from either a primary or secondary branch.

<TERM#> — **<TERM#>** specifies a term number in the range 2 through 8 to use as the trigger term.

Defaults

If no options are given, all of the sequencer storage and branch qualifiers are displayed along with the trigger term position. Upon powerup or after **tinit** initialization, the sequencer defaults to the following state:

```
tif 1 any
tsto all
telif never
```

In other words, the sequencer powers up with two sequence terms; the second sequence term is the trigger term. Any state will cause a branch from the first term to the second term; global restart is set to never and all states are stored by the analyzer.

Switching analyzer configurations from easy to complex or vice versa also resets the sequencer (that is, **tcf -c** or **tcf -e**).

Examples

To view the state of the sequencer after powerup or a **tinit**, type:

```
M> tsq
```

You will see:

```
tif 1 any
tsto all
telif never
```

Only one sequence term is used; it has a primary branch qualifier set to **any**; the trigger occurs upon branching out of this term. All states encountered are stored by the analyzer; the global restart term is disabled. With this sequencer setup, the analyzer will immediately find its trigger when a trace is initiated.

Now, let's show how the sequencer may be manipulated using the analyzer commands and the sequencer commands. Type:

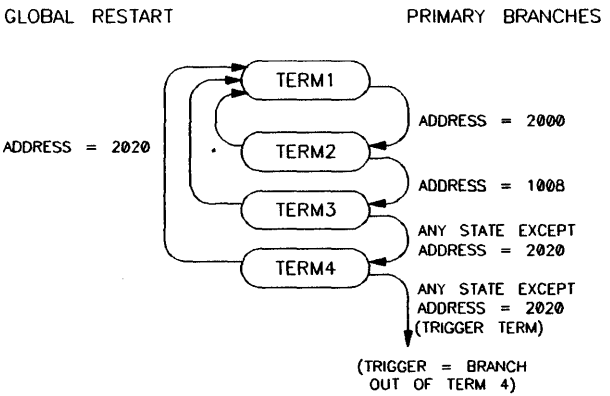
```
M> telif addr=2020
M> tsq -i 2
M> tsq -i 3
M> tsq -i 4
M> tif 1 addr=2000
M> tif 2 addr=1008
M> tif 3 addr!=2020
M> tif 4 addr!=2020
```

Since we are in the easy analyzer configuration (**tcf -e**), the **telif** command sets a global restart whenever address 2020 is encountered by the analyzer. In addition, we set the primary branch qualifier for term number 1 to the address value of 2000 hex and the primary branch qualifier for term number 2 to the address value of 1008 hex. We've also added additional terms to fill out the sequencer's limit of four terms in easy configuration. When terms are inserted, their primary branch qualifier is initially set to **tif <TERM#> any**. Now view the new sequencer arrangement by typing:

```
M> tsq
```

You will see:

```
tif 1 addr=2000
tif 2 addr=1008
tif 3 addr!=2020
tif 4 addr!=2020
tsto all
telif addr=2020
```



With this sequencer arrangement, the analyzer will first look for an address value of 2000 hex. If found, it will branch to term number 2, where it will look for an address of 1008 hex. If that value is found, the analyzer will accept any value to branch through the next two terms and then will trigger. All values found will be stored. Note that if the analyzer finds the address 2020 hex while in terms 1 through 4, it will immediately restart the sequencer at term 1; that is, it will again look for an address value of 2000 hex.

To delete a sequencer term in easy configuration, type:

```
M> tsq -d 3
```

You can verify the deletion by typing:

```
M> tsq
```

You will see:

```
tif 1 addr=2000
tif 2 addr=1008
tif 3 addr!=2020
tsto all
telif addr=2020
```

In complex configuration, the full power of the sequencer is available. There are 8 sequencer terms; any term except term 1 can be the trigger term; and each term has primary and secondary branch conditions which dictate progression to other terms in the sequence. Let's set the analyzer to complex configuration and manipulate the sequencer. Type:

```
M> tcf -c
```

This sets the analyzer to complex configuration. Now let's look at the default trace sequencer setup. Type:

```
M> tsq
```

You will see:

```
tif 1 any 2
tif 2 any 3
tif 3 any 4
tif 4 any 5
tif 5 any 6
tif 6 any 7
tif 7 any 8
tif 8 never
tsq -t 2
tsto 1 all
tsto 2 all
tsto 3 all
tsto 4 all
tsto 5 all
tsto 6 all
tsto 7 all
tsto 8 all
telif 1 never
telif 2 never
telif 3 never
telif 4 never
telif 5 never
telif 6 never
telif 7 never
telif 8 never
```

Here, the primary branch conditions are set to jump to the next term on any condition. All of the secondary branch conditions have been disabled. In addition, any state will be stored by the analyzer for each term in the sequence, and the analyzer will trigger upon entry to term number 2.

For the example below, refer to the 68000 program in Appendix A. Suppose you were having a problem with this program in that the system intermittently output MESSAGE_B when the command "A" was input and vice versa. You would like to trigger the analyzer upon such an occurrence. Type the following commands:

```
M> tpat p1 addr=2000
M> tpat p2 addr=3000
M> tpat p3 addr=206c
M> tpat p4 addr=1019
M> tpat p5 addr=1008
M> tpat p6 data=00
M> tpat p7 data=41
M> tpat p8 data=42
```

At this point, you may want to verify that all of your pattern entries are correct. Type:

```
M> tpat
```

You will see:

```
tpat p1 addr=2000
tpat p2 addr=3000
tpat p3 addr=206c
tpat p4 addr=1019
tpat p5 addr=1008
tpat p6 data=00
tpat p7 data=41
tpat p8 data=42
```

Now, set the trigger term to term 6; then set up the primary and secondary branch conditions. Type:

```
M> tsq -t 6
M> tif 1 p1 2
M> tif 2 p2 and p6 3
M> tif 3 p2 and p7 4
M> tif 4 p4 6
M> tif 5 p5 6
M> telif 3 p2 and p8 5
M> telif 4 p3 2
M> telif 5 p3 2
```

Now, verify all of the sequencer modifications by typing:

```
M> tsq
```

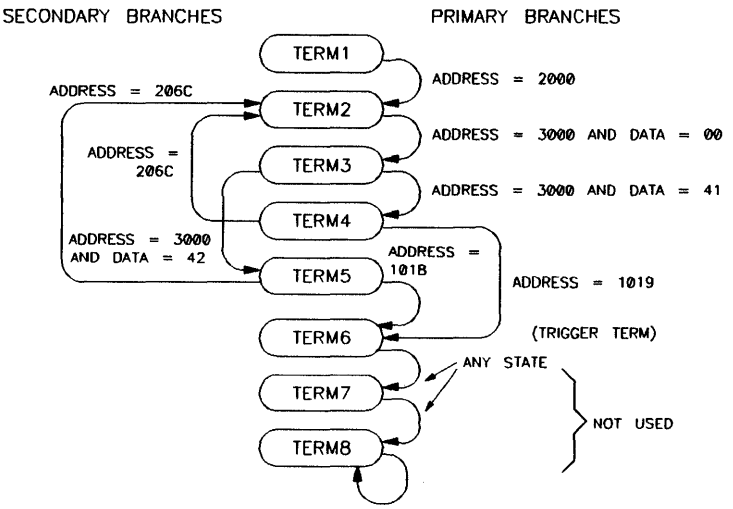
You will see:

```
tif 1 p1 2
tif 2 p2 and p6 3
tif 3 p2 and p7 4
tif 4 p4 6
tif 5 p5 6
tif 6 any 7
tif 7 any 8
tif 8 never
tsq -t 6
tsto 1 all
tsto 2 all
tsto 3 all
tsto 4 all
tsto 5 all
tsto 6 all
tsto 7 all
tsto 8 all
telif 1 never
telif 2 never
telif 3 p2 and p8 5
telif 4 p3 2
telif 5 p3 2
telif 6 never
telif 7 never
telif 8 never
```

Refer to the resulting state diagram. When the program is run, the sequencer will immediately look for the state **address=2000**. If found, the sequencer branches to term number 2, where it looks for the state where **address=3000** AND **data=00**. This will be the point where the input register is cleared by the CLEAR routine.

Note

With microprocessors such as the 68000 and the 80186 family that prefetch instructions, it is often more accurate to set up trace conditions based upon data movement resulting from an instruction rather than the instruction itself. When the data pattern is found, it is more likely that the instruction actually executed. Such methods must be used with care; in some programs several different routines may execute the same data movement.



Next, the sequencer will advance to term number 3, where it will simultaneously search for the pattern **address=3000** AND **data=41** and the pattern **address=3000** AND **data=42**. Essentially, you are now looking for command input. If **address=3000** AND **data=41** is found, command "A" has been input. To search for incorrect processing, the sequencer now jumps to term number 4 and begins looking for the pattern **address=1019**, meaning that it is searching for an access to the MESSAGE_B data area. Conversely, if **address=3000** AND **data=42** are found, the sequencer jumps to term number 5 and looks for the pattern **address=1008**, which is the MESSAGE_A data space. In either case, if the pattern **address=206c** is found first, the sequencer restarts at term 1; no error in processing was found.

If either of the **tif** patterns set up in term number 4 or 5 are satisfied, the sequencer branches to term number 6, which is the trigger term. Triggering of the analyzer occurs immediately upon entry to term number 6. In other words, if the incorrect message area is accessed for the command value input, the analyzer will trigger. Note that all states are stored in each sequence term; this will leave a record of the events leading up to the trigger state (it would be wise, however, to center the trigger in the trace memory with the command **tp c**).

Related Commands

tcf (defines whether analyzer is operated in complex configuration or easy configuration)

telif (sets global restart qualifier in easy configuration; secondary branch qualifier in complex configuration)

tg (defines the trigger qualifier)

tif (sets the primary branch qualifier in both easy and complex configuration)

tsto (defines the analyzer global storage qualifier)

Notes

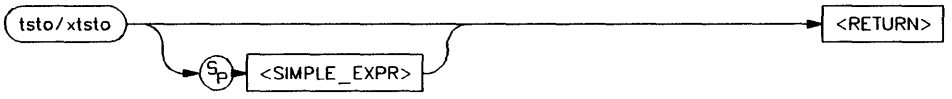
tsto,xtsto

Summary

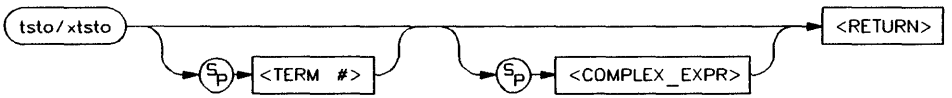
Specify analyzer trace storage qualifiers

Syntax

EASY CONFIGURATION



COMPLEX CONFIGURATION



Function

The **tsto** (**xtsto**) command allows you to specify a trace storage qualifier for the emulation (external) analyzers. The expression parameter, whether **<SIMPLE_EXPR>** or **<COMPLEX_EXPR>**, specifies the type of data to be stored by the analyzer.

If the analyzer is in easy configuration (**tcf -e**), the expression is specified by **<SIMPLE_EXPR>** and this serves as a global storage qualifier. In other words, the same expression is used as a storage qualifier regardless of the current sequencer state.

If the analyzer is in complex configuration (**tcf -c**), the expression is specified by **<COMPLEX_EXPR>** and may be assigned to a sequencer state with the **<TERM#>** parameter. When an expression is assigned to a specific term number, the analyzer will only store states corresponding to the given expression when at the given sequencer level. If no **<TERM#>** is given, the associated expression is defined as global; the analyzer stores states satisfying the expression regardless of the sequencer level.

Note

Remember that the analyzer only stores states for a given sequence term which satisfy the **tsto** qualifier for that term **while at that sequencer level**. If you specify storage of items in a particular term that occur **after** that term has been satisfied, the sequencer will no longer be at that level and therefore won't store the states you specified.

Parameters (Easy Configuration)

<SIMPLE_EXPR>

<SIMPLE_EXPR> lets you directly specify an analyzer expression to use as a storage qualifier. For example, **<SIMPLE_EXPR>** might consist of the expression **addr=2000**. For detailed information on specification of simple expressions, refer to the expression syntax pages.

Parameters (Complex Configuration)

<TERM#>

<TERM#> lets you specify a sequencer term number to associate with the given **<COMPLEX_EXPR>**. When you associate a term number with a complex expression, that expression is only used as a storage qualifier at the sequencer level specified by the term number. If you specify **<TERM#>** without an expression, the complex expression currently associated with that term number is displayed. If you specify an expression without including a **<TERM#>**, the expression is used as a global storage qualifier; that is, the storage qualifiers of all eight sequence terms are set to the same value as the global storage qualifier you specified.

Note

If you've specified a global storage qualifier, you can override any of the sequence term storage qualifiers by specifying the term number along with the new qualifier. For example, you might specify a global storage qualifier of **tsto any**; you could override this for term 3 by specifying **tsto 3 none**.

<COMPLEX_EXPR>

<COMPLEX_EXPR> allows you to specify complicated analyzer expressions made up of relationships between simple analyzer expressions. When you create a complex expression, you must first assign pattern names (**p1-p8**) to simple expressions using the **tpat** command. You then use the pattern names and relational operators to create complex expressions. For example, if you wish to store only the states where **address=2000** and **data=20** or the states **address=2000** and **data=42**, you would use the following commands:

U> **tpat p1 addr=2000 and data=20**
U> **tpat p2 addr=2000 and data=42**
U> **tsto p1 | p2**

The "|" symbol represents an intra-set OR operator. For more information on complex expressions, operators, and pattern sets, refer to the expression syntax pages in this manual.

Defaults

If no parameters are given, the current trace storage qualifier settings are displayed. Upon powerup or after **tinit** initialization, the trace storage qualifier defaults to **tsto all**. Using the **tcf** command to switch from complex configuration to easy configuration or vice versa will also reset the storage qualifier to **tsto all**.

Examples

In the following example, we'll look at a complex trace specification to store only certain data moved by the 68000 sample program (see Appendix A).

Specifically, we want the analyzer to store only the received command (input at address location 3000 hex) and the written output message (locations 4000 through 4011 hex). First, let's initialize the analyzer and set the trace configuration to complex by typing:

```
M> tinit
M> tcf -c
```

Next, we'll set up a label to be used in qualifying the data reads and writes. This label, called **lowerdata**, will overlap the analyzer's pre-defined **data** label. Type:

```
M> tlb lowerdata 40..47
```

Now we need to set up some patterns for the sequencer and storage qualifiers. We would like to set up the sequencer so the analyzer must recognize the program start at address 2000 hex, then an access to address 3000 with data not equal to a null (00 hex), then trigger on address 4000 hex with data not equal to a null (00 hex). We also want to be able to store data during accesses to the address range 4000 through 4011 hex with data not equal to zero.

To do this, we'll identify address patterns at 2000, 3000, and 4000 hex, a data pattern not equal to zero, and a range pattern encompassing the addresses 4000 through 4011 hex. Type:

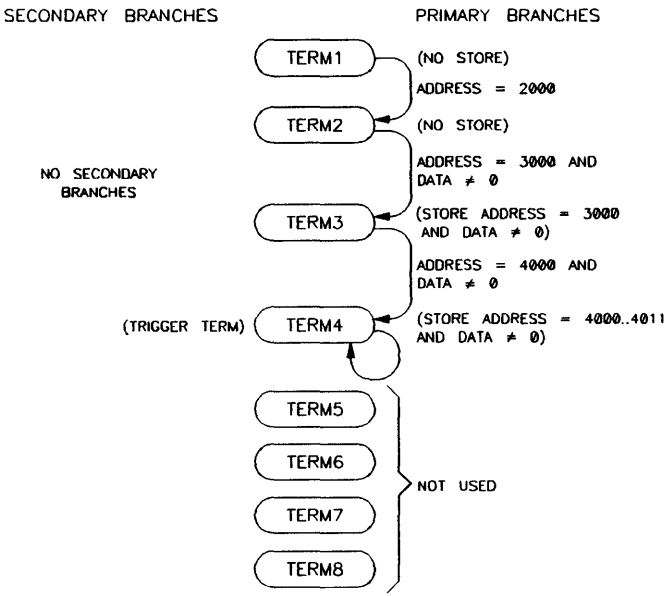
```
M> tpat p1 addr=2000
M> tpat p2 addr=3000
M> tpat p3 addr=4000
M> tpat p5 lowerdata!=00
M> trng addr=4000..4011
```

Now you can set up the sequencer. Type:

```
M> tif 1 p1 2
M> tif 2 p2 and p5 3
M> tif 3 p3 and p5 4
M> tif 4 never
M> tsq -t 4
```

If you reference these commands with the patterns above, you'll notice that the sequencer transitions from term 1 to term 2 when the address bus is equal to 2000 hex. It transitions between terms 2 and 3 when address equals 3000 but data is not equal to zero (that is, a "command" has been input). Finally, the sequencer transitions

to term 4 (the trigger term) when address equals 4000 but data does not equal zero (the message write sequence starts).



Next, set up the storage qualifiers by typing:

```
M> tsto 1 none
M> tsto 2 none
M> tsto 3 p2 and p5
M> tsto 4 r and p5
```

While the sequencer is in states 1 and 2, no data will be stored by the analyzer. When term 3 is reached, the analyzer will store all data corresponding to address 3000 AND data not equal to zero. Finally, all of the output message writes will be stored since they are in the range 4000 through 4011 hex with data not equal to zero. You can verify all of these items by typing:

```
M> tsq
```

You will see the following displayed:

```
tif 1 p1 2
tif 2 p2 and p5 3
tif 3 p3 and p5 4
tif 4 never
tif 5 any 6
tif 6 any 7
tif 7 any 8
tif 8 never
tsq -t 4
tsto 1 none
tsto 2 none
tsto 3 p2 and p5
tsto 4 r and p5
tsto 5 all
tsto 6 all
tsto 7 all
tsto 8 all
telif 1 never
telif 2 never
telif 3 never
telif 4 never
telif 5 never
telif 6 never
telif 7 never
telif 8 never
```

Now, set up the trace display format so it will show only the information of interest by typing:

```
M> tf addr,H lowerdata,A
```

Now you can start the measurement. Type:

```
M> t
```

```
Emulation trace started
```

This begins the trace. Now start the program by typing:

```
M> r 2000
```

Input one of the valid "commands" by typing:

```
U> m 3000=41
```

The emulation processor will now halt. You can halt the trace by typing:

```
h> th
```

```
Emulation trace halted
```

To display the stored analyzer information, type:

```
h> tl -2..17
```

You will see:

```
 Line   addr,H   lowerdata,A
-----   ------   -----------
   -2   002000   $
   -1   003000   A
    0   004000   T
    1   004001   H
    2   004002   I
    3   004003   S
    4   004004   .
    5   004005   I
    6   004006   S
    7   004007   .
    8   004008   M
    9   004009   E
   10   00400A   S
   11   00400B   S
   12   00400C   A
   13   00400D   G
   14   00400E   E
   15   00400F   .
   16   004010   A
   17
```

The program start address has been stored, along with the command read in by the program and the data written by the message output routines.

Related Commands

tcf (used to specify whether the analyzer is in easy configuration or complex configuration)

telif (used to specify a global restart qualifier in easy configuration; specifies a secondary branch qualifier for each sequencer level in complex configuration)

tg (used to specify a trigger condition in either easy configuration or complex configuration; overrides the current sequencer specification. Note that **tg** does not affect **tsto**; therefore, the current **tsto** specifications remain in effect whenever a **tg** command is entered)

tif (used to specify a primary branch qualifier in either analyzer configuration)

tpat (used to assign pattern names to simple analyzer expressions for use in constructing complex analyzer expressions; these expressions can be used in specifying storage qualifiers for the **tsto** command)

trng (used to specify a range of values of a set of analyzer inputs; this range information can be used in constructing complex configuration qualifiers for the **tsto** command)

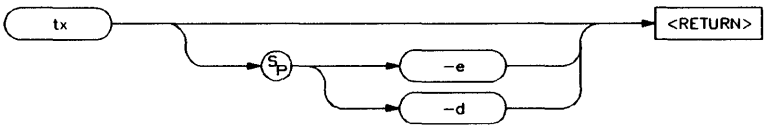
tsq (used to manipulate the trace sequencer)

tx,xtx

Summary

Set analyzer to trace on receipt of CMB /EXECUTE

Syntax



Function

The **tx** command allows you to specify that the analyzer will begin a measurement when the CMB /EXECUTE line is asserted.

If **tx -e** is given, enabling measurement on execute, the CMB trigger is immediately driven true upon receiving the /EXECUTE signal. If the analyzer is not driving either trig1 or trig2, it is then started. The CMB trigger is then disabled and the HP 64700 waits for all other participants in the measurement to release the CMB trigger. When the last instrument releases the CMB trigger, the trigger will go false; at this point any analyzers driving trig1 or trig2 will be started.

Parameters

-e	If you specify the **-e** option, this emulator will start an analysis measurement upon receiving the CMB /EXECUTE signal.
-d	If you specify the **-d** option, the emulation analyzer will NOT start an analyzer measurement upon receiving the CMB /EXECUTE signal.

Defaults

If no options are specified, the current state of **tx** enable/disable is displayed. Upon powerup or after a **tinit**, the system defaults to **tx -e**.

Examples

You may want to set up a CMB measurement such that this emulator starts running and an analyzer measurement begins at address location 2000 hex whenever the CMB /EXECUTE pulse is received. Type the following commands:

```
M> cmbt -d none
M> tx -e
M> tx
```

You will see:

```
tx -e     # start a measurement on the execute signal

M> tg addr=2000
M> rx 2000
```

The command **cmbt -d none** ensures that neither the trig1 or trig2 signals will be driving the CMB trigger line when the CMB /EXECUTE pulse is received; thus, the measurement will start immediately. Next, we enable trace on execute, and also execute a **tx** command with no parameters to verify that it is enabled. A trigger and a run at execute parameter is picked. Note that a **cmb -e** is not given; **rx** effectively accomplishes the same thing.

To verify the powerup default state of **tx**, simply type:

```
M> tx
```

You will see:

```
tx -d     # ignore the execute signal
```

Related Commands

cmbt (specifies whether the CMB trigger signal is driven or received by the internal trig1 and trig2 signals)

tarm (specifies the arm condition for the analyzer)

tg (specifies a trigger condition for the analyzer)

ver

Summary

Display Terminal Interface software version number

Syntax

Function

The **ver** command instructs the emulator to return the current emulator Terminal Interface software version numbers. You should use this command when you need to know the version number of your emulator Terminal Interface software to compare it to the *Firmware/Software Compatibility Note* for the HP 64700 PC Interface or Softkey Interface software versions.

Parameters

None.

Defaults

Not applicable.

Examples

To determine the current emulator Terminal Interface software version numbers, type:

```
M> ver
```

The system returns a display similar to the following:

```
                Copyright (c) Hewlett-Packard Co.  1987
All Rights Reserved.  Reproduction, adaptation, or translation without prior
written permission is prohibited, except as allowed under copyright laws.

  HP64700 Series Emulation System
    Version:   A.00.00 20Nov87

  HP64742 Motorola 68000 emulator
    Version:   A.00.00 20Nov87
    Speed:     12.5 MHz
    Memory:    126 KBytes

  HP64740 Emulation Analyzer
   Version:   A.00.00 20Nov87
```

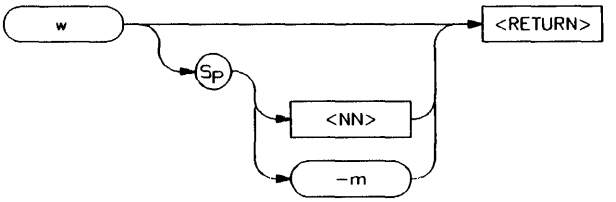
Related Commands None.

w

Summary

Wait for specified event

Syntax



Function

The w command is used to program automatic waits into macros, repeats, and command files. Normal operation is to wait for any keystroke before executing the next operation; optionally, the wait can be programmed for a specific time period or for completion of a measurement in process (such as a trace).

Parameters

<NN>	Wait for NN number of seconds before proceeding.
-m	Wait for completion of the current measurement before proceeding.

Defaults

Wait for any keystroke on the command port before proceeding.

Examples

To cause the emulator to wait for any keystroke before proceeding to the next command, type:

```
U> w
```

You might use this in a situation where you wish the operator to make a judgement regarding some other condition before proceeding with the next measurement. For example, if some LEDs in the target system should reach a certain state before a measurement is made, use the basic form of the wait command (**w**), which will allow the operator to verify that the LEDs have reached the proper state; then proceed with the next command by pressing any key.

To cause the emulator to wait for 32 seconds or for any keystroke, type:

```
U> w 32
```

This might be used where you know the desired system state will be reached in a definite amount of time (or should be reached within that time).

To have the emulator wait until another measurement is completed or for any keystroke entry, type:

```
U> w -m
```

Note that the above examples, taken exactly as shown, don't provide you with a useful function -- they are provided only to show correct examples of command line syntax. To use the wait command effectively, it should be applied within macros, repeat commands, or command files. Refer to the **rep** and **mac** commands for further examples.

Related Commands

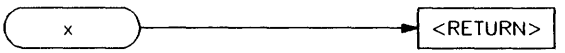
None.

x

Summary

Start synchronous CMB execution

Syntax



Function

The **x** command allows you to initiate a synchronous CMB (Coordinated Measurement Bus) measurement execution.

When **x** is performed, the CMB /EXECUTE line is pulsed. If **tx** (trace at execute) is enabled, an analyzer measurement will begin. If the CMB is enabled via the **cmb -e** command, a break will occur, followed by a run at execute as specified by the **rx** command.

The **x** command is available whether CMB and trace at execute are enabled or not. Specifically, the **cmb** and **tx** commands control how this HP 64700 emulator will respond when an /EXECUTE or READY is detected. The **x** command only controls when this emulator will issue an /EXECUTE signal.

Parameters

None.

Defaults

Does not apply.

Examples

To initiate a synchronous CMB measurement and have this HP 64700 emulator participate in the measurement, type the following commands:

```
M> rx 2000
M> tcf -e
M> tg addr=2000
M> tx
M> x
```

This enables the CMB and sets the run at execute address to 2000. The analyzer trigger is also set to 2000 hex and trace at execute is enabled. Finally, the **x** command is issued, initiating the coordinated execution. All other emulators connected to the CMB will respond as defined by their **rx, tx,** and **cmb** commands.

Related Commands

cmb (used to enable or disable interaction with the CMB)

rx (used to specify an address to start a program run when the /EXECUTE pulse is received from the CMB)

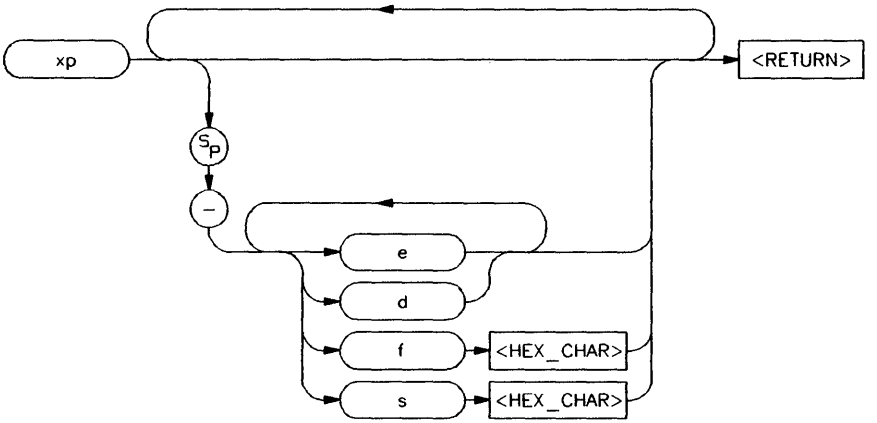
tx (used to specify that an analyzer measurement should begin when the /EXECUTE pulse is received from the CMB)

xp

Summary

Enable/disable transparent mode

Syntax



Function

The **xp** command allows you to enable or disable transparent mode and set the escape character used to temporarily escape transparent mode.

When you enable transparent mode with **xp -e**, information entered at one port is copied to the other port, that is, the HP 64700 emulator becomes a transparent data link.

When you enter the escape character sequence **<CHARF> <CHARS>**, the copying of information is temporarily suspended, allowing you to send commands to the emulator. Normally, you will use this to enter the **xp -d** command to disable transparent mode.

Parameters

-e	The **-e** option enables transparent mode. Once in transparent mode, all characters received on one of the ports will be copied to the other port. To temporarily escape transparent mode and enter an HP 64700 command, type **<CHARF><CHARS>** where **<CHARF>** is defined by the **-f** option to the **xp** command and **<CHARS>** is defined by the **-s** option to the **xp** command.
-d	The **-d** option disables transparent mode. You enter **xp -d** to disable transparent mode after temporarily suspending transparent mode by typing **<CHARF><CHARS>**.
-f	The **-f** option lets you define the first escape character **<CHARF>** in a sequence of two characters for use in temporarily suspending transparent mode.
-s	The **-s** option lets you define the second escape character **<CHARS>** (in a sequence of two) for use in temporarily suspending transparent mode.
<HEX_CHAR>	**<HEX_CHAR>** is the hexadecimal value of any ASCII character; this character will be used to temporarily suspend transparent mode.

You should use a two character sequence that would not normally be sent during typical transparent mode operation. For example, if you have a workstation connected to port B in command mode and an HP LaserJet printer connected to port A, you should not define the **<CHARF><CHARS>** sequence as **01b 028**, this is the <ESC>(sequence which is used for several printer commands. You would experience problems with such a sequence; as your applications tried to send the printer commands, the emulator would alternately enable and disable transparent mode as it saw the escape sequences pass through.

Defaults

If **xp** is entered with no parameters, the current escape character definition is displayed.

The two character escape sequence is defined as **01b 067** (**<ESC>g**) after system powerup or **init** initialization.

Examples

The first example shows you how you might log on to a remote HP-UX host over a modem using transparent mode.

For personal preference reasons, you want the two character escape sequence to be **<ESC>***. Type:

```
M> xp -f 01b -s 02a
```

You can always check the status of **xp** by typing:

```
M> xp

xp -d
xp -f 01b -s 02a
```

Now you must change the rate of your command port (currently at 9600 baud) to match that of the modem (currently 1200 baud). Type:

```
M> stty b 1200
```

(Now change the setting of your terminal to 9600 baud.)

To enable transparent mode, type:

```
M> xp -e
```

Assuming you have a Hayes compatible modem, check the modem and then dial the number of the host:

```
M> AT

OK

ATDT5555454

CONNECT
```

Once you've connected, login with your userid and password. Once you've completed the login, you'll see the host sign on screen.

```
Connected to yourhost.

yourhost

login: yourid
Password: yourpasswd
TERM = (2626) 2392
Terminal type is 2392

(c) Copyright 1983, 1984, 1985 Hewlett-Packard Co.
(c) Copyright 1979 The Regents of the University of Colorado, a body corporate
(c) Copyright 1979, 1980, 1983 The Regents of the University of California
(c) Copyright 1980, 1984 AT&T Technologies.  All Rights Reserved.

             The operating system is now Version 5.3.

   (Only authorized users are allowed to access this system.)

NOTE: If your keyboard is not used in 30 minutes, you will be logged off.
yourid       pty/ttyp3     Nov 29 09:20
```

Once you've completed your tasks, you can log out. Type:

```
exit
```

```
logout
```

Now, to return to the emulator command prompt, you must first enter the two character escape sequence and then disable transparent mode. Type:

```
<ESC>*xp -d
```

You will see the emulator prompt. Now return your command port to high speed mode by typing:

```
M> stty b 9600
```

(Remember to return your terminal setting to 9600 baud; otherwise, you won't be able to communicate with your emulator.)

In the second example, we look at printing a file to an HP LaserJet Series II printer using transparent mode. In this example, we'll be using a terminal emulator on an HP Vectra host to communicate with the emulator.

First, we need to change the escape sequence above. This is because the HP Laserjet uses the <ESC>* sequence as a command initiator for certain command groups. If you scanned the HP LaserJet command chart, you would see that the default escape sequence of <ESC>g is not used for HP LaserJet commands. So let's just return the escape sequence to the default value:

```
M> xp -f 01b -s 067
```

Now, if you want to print a file, you first need to enable transparent mode. You can do this selectively through your terminal emulator interface, or you could write a program which enables transparent mode, prints the requested files, and then disables transparent mode. For this example, we will use the terminal emulator. Type:

```
M> xp -e
```

Now you've got a direct connection to the printer. Remember, any characters you type at this point would be sent to the printer, which might give you unexpected results! To avoid this, exit the terminal emulator immediately, then print your file (the config.sys file on an HP Vectra in this case) by typing:

```
C:\ print config.sys
Name of list device (PRN:)  <RETURN>
```

Your file will be printed on the HP LaserJet. To stop communications with the printer, re-enter your terminal emulator and disable transparent mode by typing:

```
<ESC>gxp -d
```

You will see the emulator prompt:

```
M>
```

Related Commands

po (allows you to change the port assignments for standard input, standard output, and standard error)

stty (allows you to set data communications parameters such as baud rate and handshake type)

load (a load option, **-o**, also sets up transparent mode communications and **<ESC><CHAR>** must be used to complete the transaction)

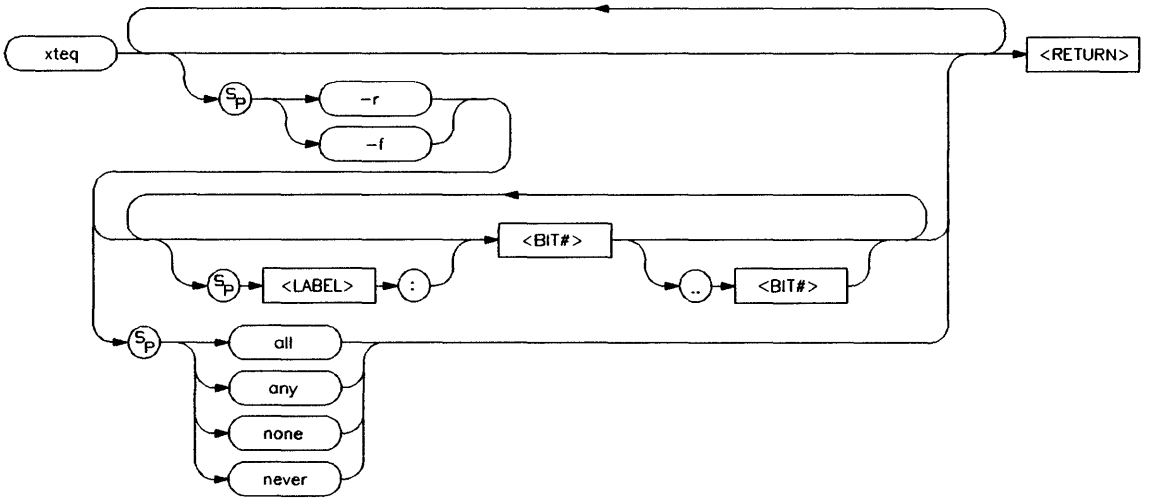
Notes

xteq

Summary

Specify external timing analyzer edge trigger

Syntax



Function

The **xteq** command allows you to specify the channels which will cause an edge trigger.

The trigger will occur following a valid duration of a pattern specified by **xtt** when a transition occurs on any of the lines specified in **xteq**. Note that **xteq** allows you to qualify the transitions to trigger only on the rising edge or the falling edge of the given input lines.

Note that the timing trace information is only accessible through the binary trace list option (**tl -b**).

Parameters

-r	If you specify **-r**, the trigger will occur on the rising edge of any signal on the input lines specified by **<BIT#>** and **<LABEL>**.
-f	If you specify **-f**, the trigger will occur on the falling edge of any signal on the input lines specified by **<BIT#>** and **<LABEL>**.
<BIT#>	**<BIT#>** specifies the bit which will cause an edge trigger. If **<BIT#>** is followed by.. and a second **<BIT#>**, they specify the range of bits which will cause a edge trigger.
<LABEL>	**<LABEL>**, when specified with a bit range (see **<BIT#>** above), specifies the bits to be used within that label which will cause a edge trigger. If **<LABEL>** is specified without a bit range, all of the bits assigned to that label will cause a edge trigger. See **xtlb** for information on specifying labels.

Note

Multiple combinations of **<LABEL>** and **<BIT#>** may be used, separated by spaces. The combinations are ORed together to form a single pattern. See the examples for details.

Note

When specifying a range of bits to use within a label, notice that the bit range specified is relative to the label, not to the input bit. For example, if you define a label named STATUS with input bits 8..11, then want to specify the least significant two bits of STATUS in a trigger specification, you can use either **STATUS:0..1** or simply the range **8..9**.

any, all	If you specify **any** or **all**, any of the external analyzer lines will cause an edge trigger for the specified edge.
none, never	If you specify **none** or **never,** none of the external analyzer lines will cause an edge trigger for the specified edge.

Defaults

If no parameters are specified, the current edge qualifier is displayed. Upon powerup or **tinit** initialization, the default setting is **xteq -r any -f any**.

Examples

Let's specify some labels for a set of status bits and a set of timer output bits to be viewed by the timing analyzer. Type:

```
M> xtlb STATUS 0..3
M> xtlb TIMER 8..11
```

Now we can set various edge qualifiers. For example, we may wish to trigger when rising edges occur on timer bits 0 through 3, when the analyzer finds the data pattern 1001 for more than 150 nanoseconds. First, set up the analyzer by typing:

```
M> xtmo -t
M> xtm -s
M> xtt STATUS=1001 > 150 n
```

Now, we can set up the edge qualifier. This can be done in two ways. Type:

```
M> xteq -r 8..11 -f none
```

To verify your choice, type:

```
M> xteq
```

You will see:

```
xteq -r 8..11 -f none
```

Or, you could type:

```
M> xteq -r TIMER:0..3 -f none
```

Again, to verify, type:

```
M> xteq
```

You will see:

```
xteq -r TIMER:0..3 -f none
```

The latter form of the command may be more useful in remembering what your motives were in assigning various bit ranges; that is, it may be helpful to remember that bits 0 through 3 were associated with the TIMER label.

If you want to trigger whenever rising or falling edges occur on any of the STATUS lines or TIMER lines, type:

```
M> xteq -r STATUS TIMER -f STATUS TIMER
```

Notice that this could alternately be specified as:

```
M> xteq -r 0..3 8..11 -f 0..3 8..11
```

Or as:

```
M> xteq -r STATUS:0..3 TIMER:0..3 -f
STATUS:0..3 TIMER:0..3
```

The last form of the command requires you to type more information. All three versions will produce the same result.

Related Commands

tlb,xtlb (specifies labels assigned to input lines for the emulation (external) analyzer)

xtgq (specifies an glitch qualifier used in conjunction with **xtt** to determine a valid trigger state)

xtm (specifies timing analyzer mode)

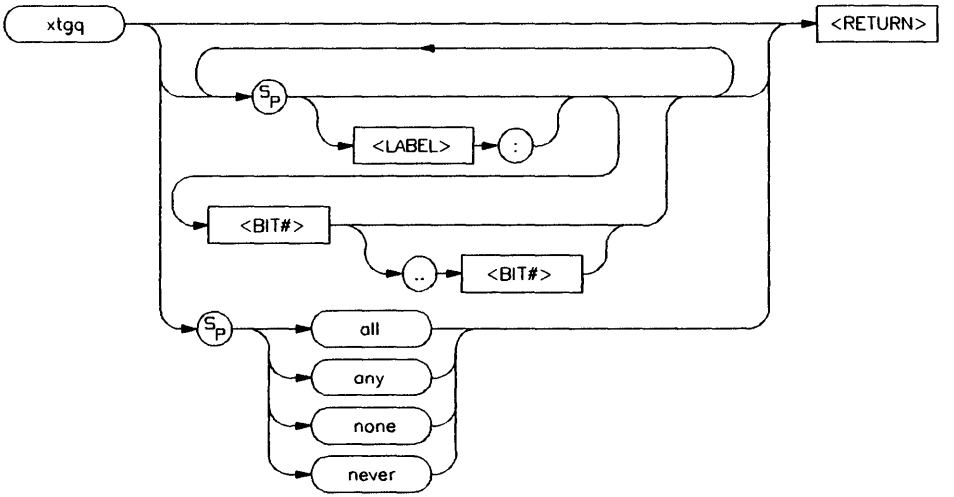
xtt (specifies timing analyzer trigger pattern and duration)

xtgq

Summary

Specify external timing analyzer glitch trigger

Syntax



Function

The **xtgq** command allows you to specify the channels which will cause a glitch trigger.

A glitch trigger will occur following a valid duration of a pattern as specified in the **xtt** command while the pattern is still present. A less than duration specified in **xtt**, or a timing mode other than **xtm -g** will cause the **xtgq** command to be ignored.

You might use this command to look for glitch occurrences related to a specific bit pattern.

Note that the timing information is only accessible through the binary trace list option (**tl -b**).

Parameters

<BIT#>	**<BIT#>** specifies the bit which will cause a glitch trigger. If **<BIT#>** is followed by.. and a second **<BIT#>**, they specify the range of bits which will cause a glitch trigger.
<LABEL>	**<LABEL>**, when specified with a bit range (see **<BIT#>** above), specifies the bits to be used within that label which will cause a glitch trigger. If **<LABEL>** is specified without a bit range, all of the bits assigned to that label will cause a glitch trigger. See **xtlb** for information on specifying labels.

Note 

Multiple combinations of **<LABEL>** and **<BIT#>** may be used, separated by spaces. The combinations are ORed together to form a single pattern. See the examples for details.

Note 

When specifying bit positions within a particular label, notice that the bit position specified are relative to the label and not the given analyzer input line. For example, if you define a label named DATA with the input bit range 8 through 15, then want to specify the two least significant bits of DATA in a trigger qualifier, you can either specify **DATA:0..1** or simply the range **8..9**.

---

any, all	If you specify **any** or **all**, any of the external analyzer lines will cause a glitch trigger.
none, never	If you specify **none** or **never,** none of the external analyzer lines will cause a glitch trigger.

Defaults

If no parameters are specified, the current glitch qualifier is displayed. Upon powerup or **tinit** initialization, the default setting is **xtgq none**.

Examples

Let's specify some labels for a set of data bits and a set of control bits to be viewed by the timing analyzer. Type:

```
M> xtlb DATA 0..7
M> xtlb FC 8..10
```

Now we can set various glitch qualifiers. For example, we may wish to trigger when glitches occur on data bits 0 through 4, when the analyzer finds the data pattern 01101110 for more than three milliseconds. First, set up the analyzer by typing:

```
M> xtmo -t
M> xtm -g
M> xtt DATA=01101110Y > 3 m
```

Now, we can set up the glitch qualifier. This can be done in two ways. Type:

```
M> xtgq 0..4
```

To verify your choice, type:

```
M> xtgq
```

You will see:

```
xtgq 0..4
```

Or, you could type:

```
M> xtgq DATA:0..4
```

Again, to verify, type:

```
M> xtgq
```

You will see:

```
xtgq DATA:0..4
```

The latter form of the command may be more useful in remembering what your motives were in assigning various bit ranges; that is, it may be helpful to remember that bits 0 through 4 were associated with the DATA label.

If you want to trigger whenever glitches occur on any of the DATA lines or on the FC lines, type:

```
M> xtgq DATA FC
```

Notice that this could alternately be specified as:

```
M> xtgq 0..10
```

Or as:

```
M> xtgq DATA:0..7 FC:0..3
```

The last form of the command requires you to type more information. All three versions will produce the same result.

Related Commands

tlb,xtlb (specifies labels assigned to input lines for the emulation (external) analyzer)

xteq (specifies an edge qualifier used in conjunction with **xtt** to determine a valid trigger state)

xtm (specifies timing analyzer mode; must be in mode **xtm -g** for **xtgq** use)

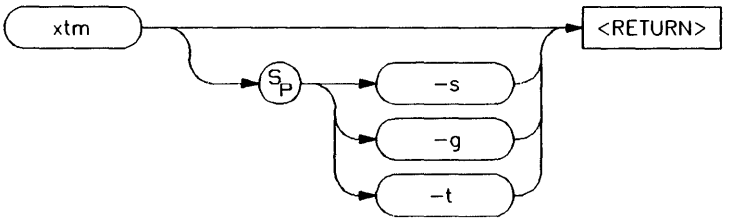
xtt (specifies timing analyzer trigger pattern and duration)

xtm

Summary

Specify external timing analyzer mode

Syntax



Function

The **xtm** command allows you to specify the mode of operation for the timing analyzer.

This command is only available if the HP 64700 emulator is equipped with the external state/timing analyzer option.

Parameters

-s

If **-s** is specified, the timing analyzer is in standard mode and samples data at the period selected by **xtsp**; up to 1024 samples can be stored during a single trace.

-g

If **-g** is specified, the timing analyzer is operated in standard mode with glitch detection added. Again, the sample rate is selected by **xtsp**. When glitch mode is selected, the maximum number of samples per trace is reduced to 512.

-t

When **-t** is specified, the timing analyzer is operated in transitional mode. Data is only stored when an input transition is detected. For the analyzer to record these transitions accurately, some trace memory must be dedicated to storing the delta time between transitions, so the number of state transitions that can be stored is reduced to a maximum of 512.

Defaults

If no parameters are supplied, the current mode setting for the timing analyzer is displayed. Upon powerup or **tinit**, the timing analyzer mode is set to **xtm -t**.

Examples

To set up the external analyzer as an independent timing analyzer in transitional mode, type:

```
M> xtmo -t
M> xtm -t
```

Related Commands

xtmo (specifies whether to use the external analyzer as a separate state analyzer, separate timing analyzer, or append the lines to the emulation analyzer)

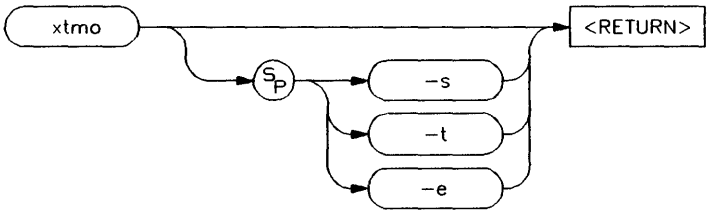
xtsp (defines the timing sample period)

xtmo

Summary

Specify external analyzer mode

Syntax



Function

The **xtmo** command allows you to specify the mode of operation for the external analyzer. The analyzer can be configured to run as an independent state or timing analyzer; or, the external analyzer can be associated with the emulation analyzer to synchronize measurements made by the two analyzers.

Note

If the emulation and external analyzers are clocking data off of the same clock, the setup/hold times of the data on the external analyzer probe inputs may not be met properly. The timing relationship between a target system processor signal and the setup/hold time of the external probe signals must be specified for each emulator. This is because each emulator has unique circuitry that generates the emulation analyzer clock and each processor has different timing requirements. Therefore, each emulator must specify the setup/hold time requirements of the external probe inputs with respect to a target processor signal.

If the external analyzer has been associated with the internal analyzer with the **xtmo -e** command, and trace specifications have been defined referencing lines present on the external analyzer, the analyzer cannot be reconfigured as an independent state or timing analyzer with the **xtmo -s** or **xtmo -t** commands until the trace specifications referencing the external analyzer lines are removed.

If the external analyzer is in the independent state or timing mode, and an **xtmo -e** command is issued to append it to the emulation analyzer, the trace specifications for the external analyzer lines are reinitialized.

Parameters

-s	If you specify the **-s** parameter, the external analyzer acts as an independent state analyzer.
-t	If you specify the **-t** parameter, the external analyzer acts as an independent timing analyzer.
-e	If you specify the **-e** parameter, the external analyzer is appended to the emulation analyzer.

Defaults

If no parameters are specified, the current operation mode of the external analyzer is displayed. Upon powerup, the default operation mode is **xtmo -e**.

Examples

To display the current operation mode of the external analyzer, type:

```
M> xtmo
```

You will see:

```
xtmo -e
```

If you want the external analyzer to function as an independent state analyzer, type:

```
M> xtmo -s
```

Related Commands

bnct (specifies whether trig1 and/or trig2 are to be driven or received by the rear panel BNC connector)

cmbt (specifies whether the trig1 and/or trig2 signals are to be driven or received by the CMB trigger line)

tarm (specifies the arm condition for the analyzer)

tgout (specifies whether or not the trig1 and/or trig2 signals are to be driven when the analyzer finds its trigger)

tx (specifies that the analyzer is to commence a trace upon receiving the CMB execute pulse)

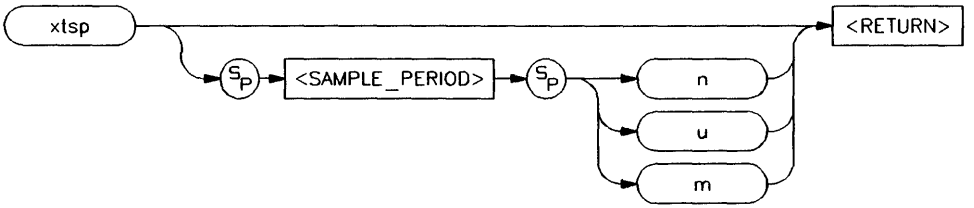
Notes

xtsp

Summary

Define external timing analyzer sample period

Syntax



Function

The **xtsp** command allows you to define the sample period for timing analyzer measurements.

Larger sample periods enable coverage of more events; however, there is the danger that some transitions may be missed if they change during the sample period. Conversely, small sample periods virtually guarantee recording of all transitions but allow the measurement of only a small total number of events in time.

Parameters

<SAMPLE_PERIOD>

<SAMPLE_PERIOD>, along with the **n,u** or **m** parameters, defines the sample period for the analyzer. This is an a integer value; the valid range for **<SAMPLE_PERIOD>** is between 10 ns and 50 ms in a 1,2,5 sequence (that is, 10 ns, 20 ns, 50 ns,..., 50 ms) for standard timing modes. For glitch mode valid periods are between 20 ns and 50 ms in the same step sequence. For transitional timing mode, the only valid sample period is 10 ns.

n	The **n** suffix indicates that the given sample period is in nanoseconds.
u	The **u** suffix indicates that the given sample period is in microseconds.
m	The **m** suffix indicates that the given sample period is in millseconds.

Defaults

If no parameters are given, the current setting of the sample period is displayed. Upon powerup or **tinit** initialization, the sample period setting is **xtsp 10 n**.

Examples

To set the timing analyzer sample period to 50 ns type:

```
M> xtm 50 n
```

To set the sample period to 200 us type:

```
M> xtm 200 u
```

And, to set the sample period to 50 ms, type:

```
M> xtm 50 m
```

You can display the current sample period setting after initialization by typing:

```
M> tinit
M> xtm
```

You will see:

```
xtm 20 n
```

Related Commands

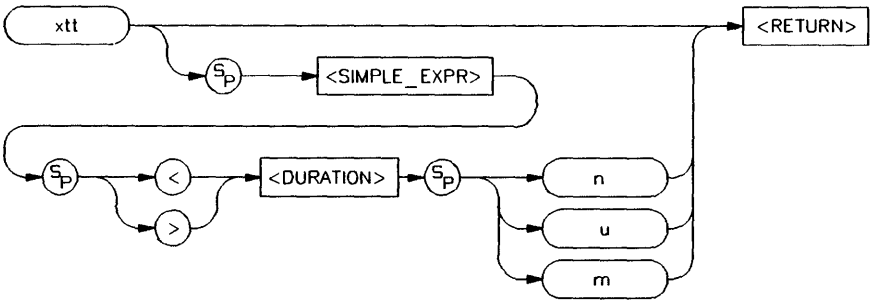
xtm (defines the timing analyzer run mode; if mode is **xtm -s** or **xtm -g**, then **xtsp** defines the amount of time between samples; if mode is **xtm -t**, the timing analyzer runs in transitional mode; the sample period (10 nanoseconds only) is used as a clock to measure the delta time between transitions)

xtt

Summary

Specify external timing analyzer trigger condition

Syntax



Function

The **xtt** command lets you specify the timing analyzer trigger. The trigger specification includes the trigger pattern and the duration of that pattern.

If **<SIMPLE_EXPR>** is found but **<DURATION>** is not satisfied, there is a 20 ns reset time before the analyzer will search for another pattern.

Parameters

<SIMPLE_EXPR> **<SIMPLE_EXPR>** defines a simple expression of the general form **label=pattern**. Other expressions may be supplied also. Refer to the syntax pages for **<SIMPLE_EXPR>** for complete details on the types of simple analyzer expressions that may be defined. Refer to the **tlb,xtlb** syntax pages for information on defining labels.

<DURATION>	The **<DURATION>** parameter, in conjunction with the greater than (**>**) and less than (**<**) operators, and the **n,u** and **m** designators, define a duration for which the trigger must be present to satisfy the trigger condition. **<DURATION>** is always expressed as an integer value. If **> <DURATION>** is specified, **<DURATION>** must fall within the range of 30 ns to 10 ms in 10 ns increments. The trigger will occur at the end of the specified duration. If **< <DURATION>** is specified, **<DURATION>** must fall within the range of 40 ns to 10 ms in 10 ns increments. The pattern must remain stable for at least 20 ns; the trigger will occur after the pattern changes states from the designated pattern.
n	The **n** suffix indicates that the duration is specified in nanoseconds.
u	The **u** suffix indicates that the duration is specified in microseconds.
m	The **m** suffix indicates that the duration is specified in milliseconds.

Defaults

If no parameters are specified, the current timing analyzer trigger expression and duration are displayed. Upon powerup or **tinit** initialization, the timing trigger is set to **xtt any**.

Examples

To define two external analyzer labels named ATN and DATA, then trigger on a certain pattern of those signals with a duration greater than 8 milliseconds, type the following commands:

```
M> xtmo -t
M> xtm -s
M> xtlb ATN 8
M> xtlb DATA 0..7
M> xtt ATN=1 and DATA=1XXXXXXXY > 8 m
M> xt
```

Related Commands

xteq (specifies that certain timing channels will qualify the trace trigger specified by **xtt**; the pattern and duration are specified by **xtt**, the trigger occurs when the signal transition specified by **xteq** occurs)

xtgq (specifies a glitch qualifier for **xtt**; the trigger occurs after the pattern and duration specified by **xtt** is satisfied when the glitch specified by **xtgq** occurs)

xtlb (defines labels for external analyzer input lines)

xtm (sets the timing mode for the analyzer to standard, glitch, or transitional)

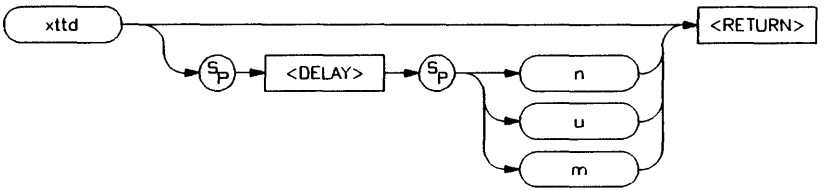
Notes

xttd

Summary

Specify external timing analyzer trigger delay

Syntax



Function

The **xttd** command allows you to specify the amount of time to delay the timing analyzer trigger after a valid trigger condition has occurred.

Parameters

<DELAY>	**<DELAY>**, along with the **n,u** or **m** parameters, defines the trigger delay period for the analyzer. This is an a integer value; the valid range for **<DELAY>** is between 0 and 10 ms in 10 ns increments.
n	The **n** suffix indicates that the given delay is in nanoseconds.
u	The **u** suffix indicates that the given delay is in microseconds.
m	The **m** suffix indicates that the given delay is in milliseconds.

Defaults

If no parameters are given, the current setting of the delay is displayed. Upon powerup or **tinit** initialization, the delay setting is **xttd 0**.

Examples

To set the timing analyzer delay to 30 ns type:

```
M> xttd 30 n
```

To set the delay to 10 ms type:

```
M> xttd 10 m
```

Note that the delay period must be specified as an integer value; real number values are not accepted (a syntax error message will be displayed).

And, to set the delay to 3 ms, type:

```
M> xttd 3 m
```

You can display the current delay setting after initialization by typing:

```
M> tinit
M> xttd
```

You will see:

```
xttd 20 n
```

Related Commands

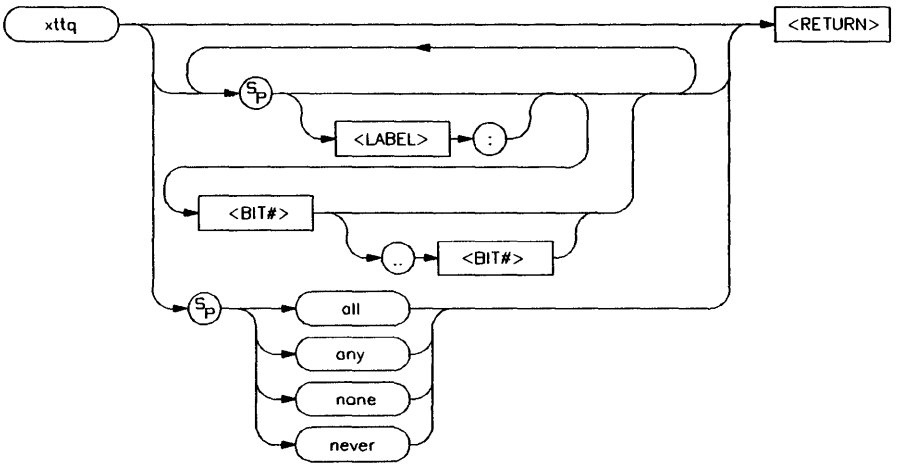
xtt (specifies the timing analyzer trigger pattern and duration)

xttq

Summary

Specify external timing analyzer transition trigger

Syntax



Function

The **xttq** command allows you to specify the channels which will cause a transition record when the timing analyzer mode is set to transitional (**xtm -t**).

Parameters

<BIT#>	**<BIT#>** specifies the bit which will cause a timing transition record. If **<BIT#>** is followed by .. and a second **<BIT#>**, they specify the range of bits which will cause a timing transition.

<LABEL> **<LABEL>**, when specified with a bit range (see **<BIT#>** above), specifies the bits to be used within that label which will cause a timing transition record. If **<LABEL>** is specified without a bit range, all of the bits assigned to that label will cause a timing transition record. See **xtlb** for information on specifying labels.

Note 

Multiple combinations of **<LABEL>** and **<BIT#>** may be used, separated by spaces. The combinations are ORed together to form a single pattern. See the examples for details.

Note 

When specifying bit positions within a given label using the construction **<LABEL>:<BIT#>..<BIT#>**, notice that the bit positions given are relative to the label and not to the analyzer input bit position. For example, if you define a label called **TIMER** with bit positions **8** through **11**, and if you then want to set up a qualifier using the two most significant bits of **TIMER**, you may either specify **TIMER:2..3** or simply the range **10..11**.

---

any, all — If you specify **any** or **all**, any of the external analyzer lines will cause a timing transition record.

none, never — If you specify **none** or **never**, none of the external analyzer lines will cause a timing transition record.

Defaults

If no parameters are specified, the current transition qualifier is displayed. Upon powerup or **tinit** initialization, the default setting is **xttq any**.

Examples

Let's specify some labels for a set of data bits and a set of control bits to be viewed by the timing analyzer. Type:

```
M> xtlb DATA 0..7
M> xtlb FC 8..10
```

Now we can set various transition qualifiers. For example, we may wish to store when transitions occur on data bits 0 through 4. This can be done in two ways. Type:

```
M> xttq 0..4
```

To verify your choice, type:

```
M> xttq
```

You will see:

```
xttq 0..4
```

Or, you could type:

```
M> xttq DATA:0..4
```

Again, to verify, type:

```
M> xttq
```

You will see:

```
xttq DATA:0..4
```

The latter form of the command may be more useful in remembering what your motives were in assigning various bit ranges; that is, it may be helpful to remember that bits 0 through 4 were associated with the DATA label.

If you want to store information whenever transitions occur on the DATA lines or on the FC lines, type:

```
M> xttq DATA FC
```

Notice that this could alternately be specified with either of the following:

```
M> xttq 0..10
M> xttq DATA:0..7 FC:0..2
```

The last form of the command requires you to type unneeded information. All three versions will produce the same result.

Related Commands

tlb,xtlb (specifies labels assigned to input lines for the emulation (external) analyzer)

xteq (specifies an edge qualifier used in conjunction with **xtt** to determine a valid trigger state)

xtgq (specifies a glitch qualifier used in conjunction with **xtt** to determine a valid trigger state)

xtm (specifies timing analyzer mode; must be in mode **xtt -t** (transitional mode) for **xttq** to be useful)

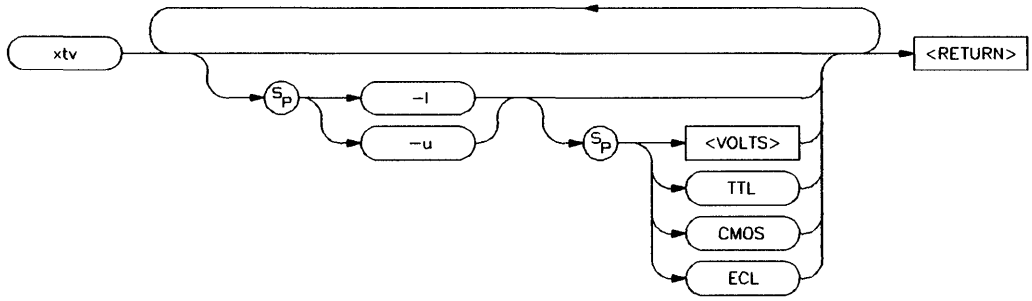
xtt (specifies timing analyzer trigger pattern and duration)

xtv

Summary

Set threshold voltages for external analyzer probes

Syntax



Function

The **xtv** command allows you to set the logic threshold voltages for the external trace probes.

Parameters

-l	The **-l** parameter indicates that the threshold voltage specified is to be used for the lower 8 bits of the analyzer probe. These are bits 0 through 7 and the **J** clock.
-u	The **-u** parameter indicates that the threshold voltage specified is to be used for the upper 8 bits of the analyzer probe. These are bits 8 through 15 and the **K** clock.
<VOLTS>	**<VOLTS>** is a number in the range of -9.9 to 9.9 which will set the specified bit range to that threshold voltage.

TTL	Specifying **TTL** sets the indicated probe's threshold voltage to a TTL (transistor-transistor-logic) level; the specific voltage used is 1.4 volts.
CMOS	Specifying **CMOS** sets the indicated probe's threshold voltage to a CMOS logic level; the specific voltage used is 2.5 volts.
ECL	Specifying **ECL** sets the indicated probe's threshold voltage to ECL levels; the specific voltage used is -1.3 volts.

Defaults

If no parameters are specified, the current threshold voltage settings are printed. Upon powerup or **tinit** initialization, the threshold voltage settings are set to **xtv -u TTL -l TTL**.

Examples

To set the threshold voltages for all external probes to ECL levels, type:

```
M> xtv -u ECL -l ECL
```

You can verify the setting by typing:

```
M> xtv
```

You will see:

```
xtv -u ECL -l ECL
```

If you need to make measurements on dual threshold voltages where the upper threshold is -1.5 volts and the lower threshold is -3.2 volts, you can double probe the signals (bit 0 and bit 8 connected to the same line, and so on through bit 15) and type the following command:

```
M> xtv -u -1.5 -l -3.2
```

Note that this will require you to carefully interpret the **xtl** trace list display, as **xtl** does not provide options for logical operations between input signals. (That is, there are no provisions for only displaying one bit and having that bit represent the output of some logical operation between two input bits.)

Related Commands

ta (allows you to view trace input signal activity; useful in verifying the correct threshold levels)

Notes

2

Expressions

This chapter includes information about these expression types:

- ANALYZER_EXPR (expressions in trace specifications)
- COMPLEX_EXPR (complex configuration expressions)
- EXPR (numeric expressions)
- SIMPLE_EXPR (easy configuration expressions)

The syntax, functional description, and related information is included for each expression type.

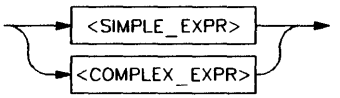
Notes

ANALYZER_EXPR

Summary

Expressions in trace specifications

Syntax



Description

Analyzer expressions are used in specifying triggers, time qualifiers, primary and secondary branch conditions, prestore qualifiers, and other analyzer setup items. There are two types of analyzer expressions, simple and complex.

In a **simple expression**, the analyzer label is related to a numeric expression within an analyzer command. These expressions are required when the analyzer is in easy configuration (**tcf -e**).

Some examples include:

```
tg addr=2000
tif 1 data=20..30
telif addr!=3000 or data!=5
```

In a **complex expression**, the relationship between an analyzer label and an expression is assigned one of 8 pattern identifiers or a range label. These patterns and the range are then used to create the actual expressions. Complex expressions are required when the analyzer is in complex configuration (**tcf -c**).

Some examples include:

First we assign a pattern name:

```
tpat p1 addr=2000
tpat p2 addr!=3000
tpat p5 data!=5
trng data=20..30
```

Then we create the actual complex expressions within the analyzer commands:

```
tg p1
tif 1 r
```

(r specifies the range defined with the **trng** command)

```
telif 1 p2 or p5 3
```

Any syntax diagram in this manual which indicates <ANALYZER_EXPR> means that a simple expression is required when the analyzer is in easy configuration, and a complex expression is required when the analyzer is in complex configuration.

Related Information

See the <SIMPLE_EXPR> and <COMPLEX_EXPR> syntax pages for complete details on each expression.

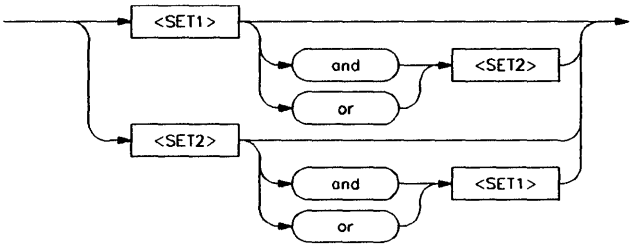
COMPLEX_EXPR

Summary

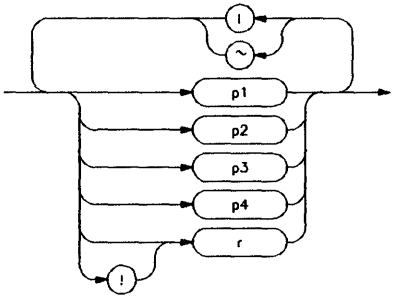
Complex configuration expressions

Syntax

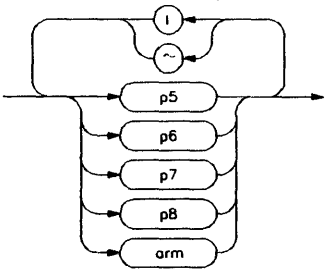
<COMPLEX_EXPR>



<SET1>
(restricted to one operator type in the set)



<SET2>
(restricted to one operator type in the set)



Description

In analyzer complex configuration (**tcf -c**) you use pattern labels, which have been assigned to various simple expressions, to form complex expressions.

Pattern Labels and Ranges

You assign pattern labels to simple expression using the **tpat** command. For example:

```
tpat p1 addr=2000
tpat p2 data!=00
tpat p3 stat=dma
tpat p4 addr=2000 and data=23
tpat p5 addr!=2105 and data!=0fc
```

You use the **trng** command to provide assign the range label:

```
trng data=42..44
```

Sets

The pattern labels, along with the range and arm specifications, are divided into two sets.

Set 1:

p1,p2,p3,p4,r,!r

Set 2:

p5,p6,p7,p8,arm

Intraset Operations

You use intraset operators to form relational expressions between members of the same set. The operators are:

~ (intraset logical NOR)

| (intraset logical OR)

The operators must remain the same throughout a given intraset expression. So, you could form the following types of intraset expressions:

```
p1~p2~r
```

(Pattern 1 NOR pattern 2 NOR range.)

```
p2 | !r
```

(Pattern 2 OR (NOT range).)

```
p5 | arm
```

(Pattern 5 OR arm.)

```
p6 ~ p8
```

(Pattern 6 NOR pattern8.)

You **cannot** use the intraset operators to form expressions between set 1 and set 2. Also, remember that the intraset operator must remain the same throughout the set. Therefore, the following examples are **invalid**:

```
p2~p3|p4
```

(This is incorrect because the operator must remain the same throughout the set.)

```
p2~p5
```

(You cannot use intraset operators for interset operations.)

Interset Operations

You use interset operators to form relational expressions between members of set 1 and set 2. The operators are:

and (interset logical AND)

or (interset logical OR)

You can then form the following types of expressions:

(set 1 expression) and (set 2 expression)

(set 1 expression) or (set 2 expression)

The order of sets does not matter:

(set 2 expression) and (set 1 expression)

Combination

You can use both the intraset and interset operators to form very powerful expressions.

```
p1~p2 and p5|arm
p3 or p6~p7~p8
```

However, you cannot repeat different sets to extend the expression. The following is **invalid:**

```
p1~p2 and p5 and p3 and p7
```

DeMorgan's Theorem and Complex Expressions

At first glance, it seems that you only have a few operators to form logical expressions. However, using the combination of the simple and complex expression operators, along with a knowledge of DeMorgan's Theorem, you can form virtually any expression you might need in setting up an analyzer specification.

DeMorgan's theorem in brief says that

A NOR B = (NOT A) AND (NOT B)

and

A NAND B = (NOT A) OR (NOT B)

The NOR function is provided as an intraset operator. However, the NAND function is not provided directly. Suppose you wanted to set up an analyzer trace of the condition

(addr=2000) NAND (data=23)

This can be done easily using the simple and complex expression capabilities. First, you would define the simple expressions as the inverse of the values you wanted to NAND:

```
tpat p1 addr!=2000
tpat p2 data!=23
```

Then you would OR these together using the intraset operators:

```
p1|p2
```

This is effectively the same as:

(NOT addr=2000) OR (NOT data=23) = (addr=2000) NAND (data=23)

If you need an intraset AND operator, you can use the same theory. Suppose you actually wanted:

(addr=2000) AND (data=23)

First, define the simple expressions as the inverse values:

```
tpat p1 addr!=2000
tpat p2 data!=23
```

Then you would NOR these together using the intraset operators:

```
p1~p2
```

This is effectively the same as:

(NOT addr=2000) NOR (NOT data=23) = (addr=2000) AND (data=23)

Related Information

See the <EXPR> syntax pages for information on numeric expression specifications. See the <SIMPLE_EXPR> syntax pages for information on the types of simple expressions that may be assigned pattern names. Also, refer to the *Emulator User's Guide* for your emulator for information on address specifications.

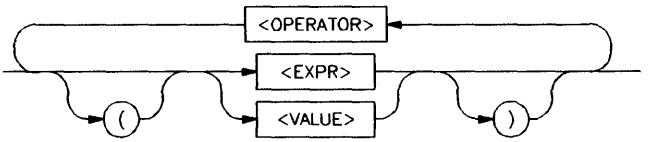
Notes

EXPR

Summary

Numeric expressions

Syntax



Description

Numeric expressions are the root of all HP 64700 Terminal Interface expression types, including analyzer expressions, address specifications, equates, and expressions you might want to calculate using the **echo** command.

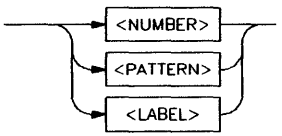
The expression capability in the Terminal Interface is very powerful; you may specify numbers in one of four different bases and use many different arithmetic and logical operators to form more complex expressions.

Terminal Interface expressions consist of other **expressions** (recursion) and **values**, which may be modified by various **operators**. You may change the precedence of operators by enclosing expressions within parentheses.

Values

Values consist of **numbers** (in one of four bases), **patterns** (hexadecimal, octal, or binary numbers that also include don't care values), **labels** (only labels pointing to other numbers or patterns, assigned by the **equ** command), and symbols.

<VALUE>



Numbers are in hexadecimal, decimal, octal, or binary. You specify the base as follows:

Y y	Binary (example: 10010y)
Q q O o	Octal (example: 377o or 377q)
T t	Decimal (example: 197T)
H h	Hexadecimal (example: 0A7fH) (Note that hexadecimal numbers starting with any one of the letter digits A-F must be prefixed with a zero; otherwise the system will return an error message)

If you do not specify a base, numbers default to hexadecimal or decimal, depending on the context.

All numbers used in equates, echo, address specification, analyzer expressions, and any other specification relating to a microprocessor address, data or status value defaults to hexadecimal.

Numbers used to specify repeat count values, such as in the sequence branch commands, trigger, step, repeat command, and so on, default to decimal.

Patterns are hexadecimal, octal, or binary numbers which include don't care digits, specified by the letters **X** or **x**. The character **?** represents a pattern of all don't care digits. For example:

1011xx11y

0A7Xh (equivalent to 000010100111xxxxy)

2x5Q (equivalent to 010xxx101y)

You will generally use patterns only in analyzer expressions. A place where you might want to use don't care values is to simulate a second range variable in complex mode specifications. For example, you might have:

trng addr=4000..4020

And you need a second range of **data** from 11 through 14 hex. Although it isn't perfect, you can simulate a second range by assigning a pattern label as follows:

tpat p1 data=00010XXXy

(This actually gives a range from 10 to 17 hex.)

Note 

Don't care values are not allowed in expressions for the **echo** command.

Labels refer to names equated to numbers, patterns, or other expressions using the **equ** command.

Operators

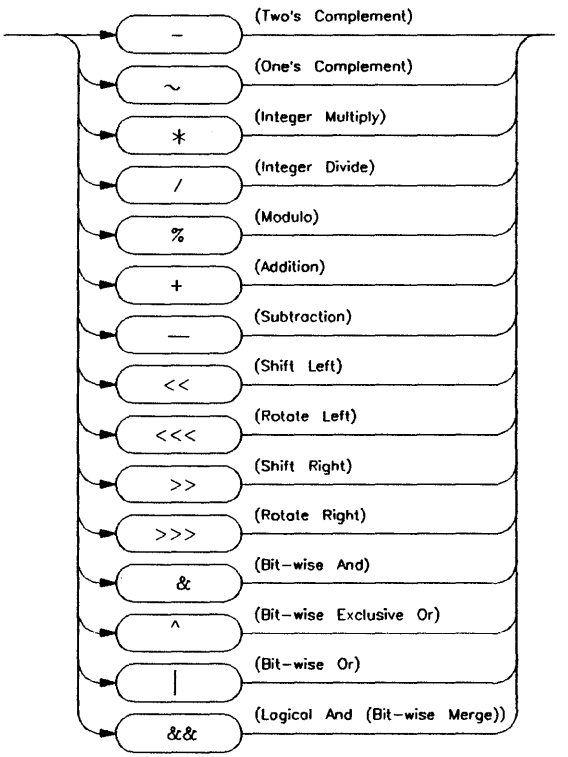
The expression capability includes a powerful set of operators, freeing you from the need to calculate expressions before entering them into other expressions. All operations are carried out on 32 bit two's complement signed integers (values which are not 32 bit will be padded out with zeros when expression evaluation occurs).

The operators are listed in the following diagram and described in order of evaluation precedence. As mentioned above, you may use parentheses in the expression to change the order of evaluation.

Note 

If your emulator supports symbols, and you are using a symbol in an expression, only the + and - operators are valid before and after the symbol. For example: **m -dm 100h+main-5**

<OPERATOR>



- ~

Unary two's complement, unary one's complement. Two's complement is not allowed on patterns containing don't care bits. This is the truth table for one's complement:

0 => 1
1 => 0
X => X

Examples:

~ 1x0y = 0x1Y

-1101Y = 0011Y

* / %	Integer multiply, integer divide, integer modulo. These operations are not allowed on patterns containing don't care bits.

Examples:

30afH*21 = 06468fH

23T%4T=3

0fa6/2 = 07d3h

\+ -Addition, subtraction. Not allowed on patterns containing don't care bits.

Examples:

03dh+03fh = 07ch

1110Y-101Y = 1001Y

<< <<< >> >>>	Shift left, rotate left, shift right, rotate right (you must specify the number of locations to shift or rotate after the operator).

Examples:

1x0Y<<1 = 1x00Y

1x0Y>>1 = 01xY

1x01Y>>>1 = 10000000000000000000000000001x0Y

0xxf0abcdH>>>4 = 0dxxf0abcH

&

This symbol (&) represents a bit-wise AND operation. The truth table resembles:

&	0	1	X
0	0	0	0
1	0	1	X
X	0	X	X

For example:

10xxy&11x1Y = 10xxY

^

This symbol (^) represents a bit-wise exclusive OR operation. The truth table resembles:

^ ^	0	1	X
0	0	1	0
1	1	0	X
X	0	X	X

For example:

10xxY ^ 11x1Y = 01xxY

|

This symbol (|) represents a bit-wise inclusive OR operation. The truth table resembles:

|	0	1	X
0	0	1	0
1	1	1	1
X	0	1	X

For example:

10xxY|11x1Y = 11x1Y

&&

This symbol (&&) represents a bit-wise merge operation. The truth table resembles:

&&	0	1	X
0	0	*	0
1	*	1	1
X	0	1	X

An overlap, indicated by a * in the merge truth table, may occur if two patterns specify different values for a pattern bit. If an overlap occurs, the first pattern's value for that bit overrides the second pattern's value.

For example:

10xxY&&11x1Y = 10x1Y

Using Expressions in Addressing and Analyzer Expressions

You can use the expression evaluation capability to form more powerful expressions for use in specifying addressing and analyzer expressions. For example, suppose you want to trigger the analyzer on the access to trap vector 13. Instead of calculating the address, since you know the base address is 080 hex and each vector is 4 address bytes, you can specify this as:

tg addr=(080h+(13T*4))

You could simplify the above even further using the equate command to assign names to some of the values. For example:

equ trapvectorbase=080h

equ trapvectorlength=4

Then:

tg addr=(trapvectorbase+(13*trapvectorlength))

Related Information

Refer to the <ANALYZER_EXPR>, <SIMPLE_EXPR>, and <COMPLEX_EXPR> pages for information on the use of expressions in forming analyzer expressions.

Refer to the **echo** and **equ** command syntax pages for information on use of expressions in expression calculation and equates.

Refer to the <ADDRESS> syntax pages in the *Emulator User's Guide* for information on use of expressions in addressing.

SIMPLE_EXPR

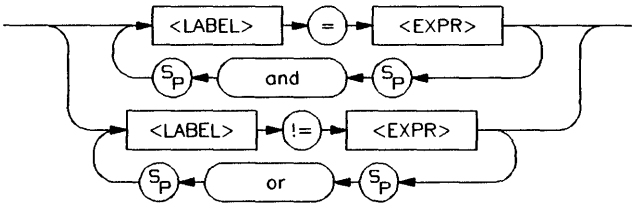
Summary

Easy configuration expressions

Syntax

EASY CONFIGURATION ONLY

EASY AND COMPLEX CONFIGURATION



Description

Easy Configuration

When the analyzer is in easy configuration (**tcf -e**), simple expressions are used to set up trace qualifiers for sequencer branches, triggers, state counting, and so on. These expressions can take the following forms:

label=expression

Examples addr=2000h

data=25h+20h

stat=0110xxxxY

label!=expression

Examples stat!=suprdata (notice that the expression can also be an equate label)

data!=00

label=expression..expression

Examples addr=4000..401

data=41..42

label!=expression..expression

Examples addr!=1000..1038

data!=00..40

Note 

No more than one simple expression can exist at any given time which is in the form of a range (expr..expr).

---

label=expression and label=expression

Examples addr=3000 and data=41

addr=start and data=00

label!=expression or label!=expression

Examples addr!=3000 or data!=41

Complex Configuration

In analyzer complex configuration (**tcf -c**), you assign each simple expression a pattern name using the **tpat** command. These pattern names are then combined to form complex expressions involving relationships between multiple simple expressions.

With the exception of these two expressions:

label=expression..expression

label!=expression..expression

all of the simple expression types can be assigned pattern names by **tpat** in complex configuration. To form ranges of expressions in complex configuration, you use the **trng** command.

Examples

```
tpat p1 addr!=3000 or data!=41

tpat p2 data=23

trng addr=1000..1038
```

(You don't need the != relation in ranges because all complex expressions provide for the logical **not** of the range specifier.)

Invalid Simple Expressions

The following simple expressions are invalid in either analyzer configuration. If you need expressions of these types, you must switch to complex configuration, assign pattern names to subparts of these expressions, then combine them using the complex expression capability.

label=expression and label!=expression

This is incorrect because you must use only the = relation with the **and** operator. To represent this, switch to complex configuration and do the following:

```
tpat p1 label=expression

tpat p5 label!=expression
```

Now, you would represent the above (incorrect) simple expression as a complex expression of the form:

p1 and p5

label!=expression or label=expression

A similar problem exists here. You must use only the != relation with the **or** operator. To represent this, switch to complex configuration and do one of the following.

tpat p1 label!=expression

tpat p2 label=expression

You would represent the above (incorrect) simple expression as a complex expression of the form:

p1 | p2

You could also do this:

tpat p1 label!=expression

tpat p5 label=expression

Represent this in complex form as:

p1 or p5

Refer to the <COMPLEX_EXPR> syntax pages for more details on forming complex expressions.

Related Information

See the <EXPR> syntax pages for information on numeric expression specifications. Also, refer to the *Emulator User's Guide* for your emulator for information on address specifications.

A

Sample Programs

- A copy of the 68000 sample program used in the examples in this manual is included, along with brief instructions for loading the program.
- An 80186 version of the same program is included. Label names and address locations of the routines vary from the 68000 version. You can modify the examples accordingly.
- Information about loading and using symbol files for HP 64700-Series Emulators that support symbols is included.
- An 8051 sample program is included. The 8051 Emulator supports symbols, so this program allows you to become familiar with using symbols.

68000 Sample Program

```
FILE: ~/68kcode/newprog  HEWLETT-PACKARD: 68000 Assembler
Sat Dec 12 11:15:27 1987    PAGE    1

LOCATION OBJECT CODE LINE     SOURCE LINE

                       1 "68000"
                       2               DATA
                       3
                       4
  000000 0000 3000     5 INPUT_POINTER  DC.L 00003000H
  000004 0000 4000     6 OUTPUT_POINTER DC.L 00004000H
                       7
  000008 5448495320    8 MESSAGE_A      ASCII     "THIS IS MESSAGE A"
  00000D 4953204D45
  000012 5353414745
  000017 2041
```

```
                          9
  000019 5448495320      10 MESSAGE_B      ASCII      "THIS IS MESSAGE B"
  00001E 4953204D45
  000023 5353414745
  000028 2042
                         11
  00002A 494E56414C      12 INVALID_INPUT ASCII      "INVALID COMMAND"
  00002F 494420434F
  000034 4D4D414E44
                         13
                         14                PROG
                         15
  000000 2479            16 INIT           MOVE.L INPUT_POINTER,A2
  000002 00000000
  000006 2679            17                MOVE.L OUTPUT_POINTER,A3
  000008 00000004
                         18
  00000C 14BC 0000       19 CLEAR          MOVE.B #00H,[A2]
                         20
  000010 1012            21 READ_INPUT     MOVE.B [A2],D0
  000012 0C00 0000       22                CMP.B #00h,D0
  000016 67F8            23                BEQ READ_INPUT
                         24
  000018 0C00 0041       25 PROCESS_COMM   CMP.B #41H,D0
  00001C 6700 000E       26                BEQ COMMAND_A
  000020 0C00 0042       27                CMP.B #42H,D0
  000024 6700 0014       28                BEQ COMMAND_B
  000028 6000 001E       29                BRA UNRECOGNIZED
                         30
  00002C 103C 0011       31 COMMAND_A      MOVE.B #11H,D0
  000030 207C            32                MOVE.L #MESSAGE_A,A0
  000032 00000008
  000036 6000 001A       33                BRA OUTPUT
  00003A 103C 0011       34 COMMAND_B      MOVE.B #11H,D0
  00003E 207C            35                MOVE.L #MESSAGE_B,A0
  000040 00000019
  000044 6000 000C       36                BRA OUTPUT
  000048 103C 000F       37 UNRECOGNIZED   MOVE.B #0FH,D0
  00004C 207C            38                MOVE.L #INVALID_INPUT,A0
  00004E 0000002A
                         39
                         40
  000052 224B            41 OUTPUT         MOVE.L A3,A1
                         42
  000054 123C 0020       43 CLEAR_OLD      MOVE.B #20H,D1

FILE: ~/68kcode/newprog  HEWLETT-PACKARD: 68000 Assembler
Sat Dec 12 11:15:27 1987    PAGE    2

LOCATION OBJECT CODE LINE     SOURCE LINE

  000058 2A4B            44                MOVE.L A3,A5
  00005A 1AFC 0000       45 CLEAR_LOOP     MOVE.B #00H,[A5]+
  00005E 0441 0001       46                SUBI #01H,D1
  000062 66F6            47                BNE CLEAR_LOOP
                         48
  000064 12D8            49 LOOP           MOVE.B [A0]+,[A1]+
  000066 0440 0001       50                SUBI #01H,D0
  00006A 66F8            51                BNE LOOP
  00006C 4EF9            52                JMP CLEAR
  00006E 0000000C
                         53
                         54
```

```
                        55                END

Errors=    0
FILE: ~/68kcode/newprog  CROSS REFERENCE TABLE          PAGE 3

LINE#    SYMBOL          TYPE      REFERENCES

   19   CLEAR             P     52
   45   CLEAR_LOOP        P     47
   43   CLEAR_OLD         P
   31   COMMAND_A         P     26
   34   COMMAND_B         P     28
   16   INIT              P
    5   INPUT_POINTER     D     16
   12   INVALID_INPUT     D     38
   49   LOOP              P     51
    8   MESSAGE_A         D     32
   10   MESSAGE_B         D     35
   41   OUTPUT            P     33,   36
    6   OUTPUT_POINTER    D     17
   25   PROCESS_COMM      P
   21   READ_INPUT        P     23
   37   UNRECOGNIZED      P     29
```

Loading the 68000 Sample Program

Set up Memory Map and the Stack Pointer

Before you load the program, you must map memory and set up the stack pointer. Here are the necessary commands:

```
M> map 1000..1fff eram
M> map 2000..2fff erom
M> map 3000..5fff eram
M> reg ssp=5000
```

Transferring Code from a Terminal in Standalone Configuration

To load a program into emulation memory using a data terminal, you must use the memory modification command.

These are the commands used to load the 68000 sample program.

```
M> m -db 1000..100f=00,00,30,00,00,00,40,00,54,48,49,53,20,49,53,20
M> m 1010..101f=4d,45,53,53,41,47,45,20,41,54,48,49,53,20,49,53
M> m 1020..102f=20,4d,45,53,53,41,47,45,20,42,49,4e,56,41,4c,49
M> m 1030..1038=44,20,43,4f,4d,4d,41,4e,44
M> m 2000..200f=24,79,00,00,10,00,26,79,00,00,10,04,14,0bc,00,00
M> m 2010..201f=10,12,0c,00,00,00,67,0f8,0c,00,00,41,67,00,00,0e
M> m 2020..202f=0c,00,00,42,67,00,00,14,60,00,00,1e,10,3c,00,11
M> m 2030..203f=20,7c,00,00,10,08,60,00,00,1a,10,3c,00,11,20,7c
M> m 2040..204f=00,00,10,19,60,00,00,0c,10,3c,00,0f,20,7c,00,00
M> m 2050..205f=10,2a,22,4b,12,3c,00,20,2a,4b,1a,0fc,00,00,04,41
M> m 2060..206f=00,01,66,0f6,12,0d8,04,40,00,01,66,0f8,4e,0f9,00,00
M> m 2070..2071=20,0c
```

Transferring Code from a Host, HP 64700 in Transparent Configuration

The method provided example assumes that you are running an HP 64000 Hosted Development System (HDS) Assembler on an HP 9000/300 or HP 9000/500 computer running the HP-UX operating system. In addition, you must have the HP 64000 **transfer** software running on your host.

If you are not using an HP 64000 HDS assembler, you may be able to adapt the methods below to load your code into the emulator.

If you are not able to transfer code from your host to the emulator using one of these methods, use the method described under the previous "Transferring Code from a Terminal in Standalone Configuration", as it will work in all cases. However, transferring code using host transfer facilities is easier and faster than modifying memory locations, especially for large programs.

1. First, you must establish communications with your host computer through the transparent mode link provided in the HP 64700. Type:

```
M> xp -s 02a
```

This sets the second escape character to *. (The first escape character remains at the HP 64700 powerup default of hex 01b, which is the ASCII <ESC> character.) The sequence "**<ESC> ***"

toggles the transparent mode software within the HP 64700 for the duration of one command (that is, any valid line of HP 64700 commands (not to exceed 254 characters) concatenated by semicolons and terminated by a <carriage return>).

1. Enable the transparent mode link by typing:

```
M> xp -e
```

If you then press <RETURN> a few times, you should see:

```
login:
login:
login:
```

This is the login prompt for an HP-UX host system. (Your prompt may differ depending on how your system manager has configured your system.)

1. Log in to your host system and start up an editor such as "vi." You should now enter the source code for the sample program shown in the listing. When finished, save the program to filename "progsamp."

Note

If you need help learning how to log in to your HP-UX host system or use other features of the system, such as editors, refer to the *HP-UX Concepts and Tutorials* guides and your HP-UX system administrator.

---

2. Assemble your code with the HP 64000 HDS Assembler using the command:

```
$ asm -oex progsamp
```

This will generate an expanded listing with cross reference table of all the symbols used. If any assembly errors were reported, re-edit your file and verify that the code was entered correctly (compare to the listing).

1. Link the program to the correct addresses using the command:

```
$ lnk
```

The system will prompt you to enter various pieces of information as follows:

```
object files  progsamp
library files <CR>
load addresses:PROG,DATA,COMN,A5
2000h,1000h,1000h
more files (y or n) <CR>
absolute file name progsamp
```

The linker will generate the absolute code required and place it in the file progsamp.X.

Now it's time to transfer your code into the emulator. Do the following:

1. Disable the transparent mode so that your terminal will talk directly to the emulator. Type:

```
$ <ESC>*xp -d
```

The "**<ESC>***" sequence temporarily toggles the transparent mode so that the emulator will accept commands; "**xp -d**" then fully disables the transparent mode.

1. Load code into the emulator by typing:

```
M> load -hbo
transfer -tb progsamp.X<ESC>* (NOTE: DO NOT
TYPE CARRIAGE RETURN!)
```

The system will respond:

```
##
M>
```

80186 Sample Program

```
FILE: cmd_rdr.S          HEWLETT-PACKARD: 80186 Assembler

LOCATION OBJECT CODE LINE     SOURCE LINE

                      1 "80186"
                      2                 ORG     500H
    0500              3 Msgs
    0500 436F6D6D61   4 Msg_A           DB      "Command A entered "
    0505 6E64204120
    050A 656E746572
    050F 656420
    0512 456E746572   5 Msg_B           DB      "Entered B command "
    0517 6564204220
    051C 636F6D6D61
    0521 6E6420
    0524 496E76616C   6 Msg_I           DB      "Invalid Command "
    0529 696420436F
    052E 6D6D616E64
    0533 20
    0534              7 End_Msgs
                      8
                      9                 ORG     400H
                     10                 ASSUME  DS:ORG,ES:ORG
                     11 ************************************************
                     12 * The following instructions initialize segment
                     13 * registers and set up the stack pointer.
                     14 ************************************************
    0400 B80000      15 Init            MOV     AX,SEG Msg_A
    0403 8ED8        16                 MOV     DS,AX
    0405 B80000      17                 MOV     AX,SEG Cmd_Input
    0408 8EC0        18                 MOV     ES,AX
    040A 8ED0        19                 MOV     SS,AX
    040C BCF906      20                 MOV     SP,OFFSET Stk
                     21 ************************************************
                     22 * Clear previous command.
                     23 ************************************************
    040F 26C6060006  24 Read_Cmd        MOV     Cmd_Input,#0
    0414 0090
                     25 ************************************************
                     26 * Read command input byte.  If no command has been
                     27 * entered, continue to scan for command input.
                     28 ************************************************
    0416 26A00006    29 Scan            MOV     AL,Cmd_Input
    041A 3C00        30                 CMP     AL,#0
    041C 74F8        31                 JE      Scan
                     32 ************************************************
                     33 * A command has been entered.  Check if it is
                     34 * command A, command B, or invalid.
                     35 ************************************************
    041E 3C41        36 Exe_Cmd         CMP     AL,#41H
    0420 7407        37                 JE      Cmd_A
    0422 3C42        38                 CMP     AL,#42H
    0424 740C        39                 JE      Cmd_B
    0426 E91200      40                 JMP     Cmd_I
```

```
                    41 ************************************************
                    42 * Command A is entered.  CX = the number of bytes in
                    43 * message A.  SI = location of the message.  Jump to
                    44 * the routine which writes the messages.
                    45 ************************************************
0429 B91200         46 Cmd_A            MOV     CX,#Msg_B-Msg_A
042C BE0005         47                  MOV     SI,OFFSET Msg_A
042F E90F00         48                  JMP     Write_Msg
                    49 ************************************************
                    50 * Command B is entered.
                    51 ************************************************
0432 B91200         52 Cmd_B            MOV     CX,#Msg_I-Msg_B
0435 BE1205         53                  MOV     SI,OFFSET Msg_B
0438 E90600         54                  JMP     Write_Msg
                    55 ************************************************
                    56 * An invalid command is entered.
                    57 ************************************************
043B B91000         58 Cmd_I            MOV     CX,#End_Msgs-Msg_I
043E BE2405         59                  MOV     SI,OFFSET Msg_I
                    60 ************************************************
                    61 * Message is written to the destination.
                    62 ************************************************
0441 8D3E0106       63 Write_Msg        LEA     DI,Msg_Dest
0445 F3A4           64                REP MOVSB
                    65 ************************************************
                    66 * The rest of the destination area is filled
                    67 * with zeros.
                    68 ************************************************
0447 C60500         69 Fill_Dest        MOV     BYTE PTR [DI],#0
044A 47             70                  INC     DI
044B 81FF2106       71                  CMP     DI,#Msg_Dest+20H
044F 75F6           72                  JNE     Fill_Dest
                    73 ************************************************
                    74 * Go back and scan for next command.
                    75 ************************************************
0451 EBBC           76                  JMP     Read_Cmd
                    77
                    78                  ORG     600H
                    79 ************************************************
                    80 * Command input byte.
                    81 ************************************************
0600                82 Cmd_Input        DBS     1
                    83 ************************************************
                    84 * Destination of the command messages.
                    85 ************************************************
0601                86 Msg_Dest         DDS     3EH
06F9                87 Stk              DWS     1       ; Stack area.
                    88                  END

Errors=    0
```

Symbol Files

Versions of HP 64700-Series emulator firmware that support symbol files can load an ASCII text file containing symbol definitions.

Three types of symbols can be defined: local, global, and user. Only local and global symbols can be loaded from a symbol file; user symbols can only be created with the **sym** command.

Global symbols are general memory references. They represent the equivalent of "GLOBAL" or "PUBLIC" variables in compiled programs.

Local symbols are grouped by "module." The primary purpose of a module is to group local symbols, but can represent any arrangement of local symbols desired. Local symbols created by a higher level language processor are defined by implementation.

A module is usually a source file name, and symbols are function or procedure names. In a symbol file, any organizational scheme can be used to manage local symbols. While the module name can be equivalent to a source file name, or some other physical or logical entity, it is not necessary. Therefore, if memory is in short supply, you can organize the "local" symbols to allow for easy deletion of old symbols, and loading of new symbols that reference locations of interest.

Address references for all symbol types are absolute addresses.

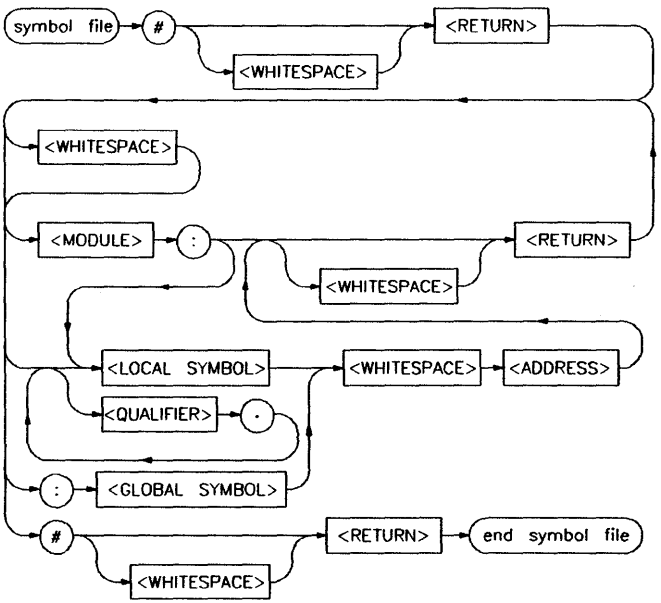
Note

The **sym** command presently applies to some HP 64700-Series emulators, and may apply to all HP 64700-Series emulators in the future. If you are using an emulator that does not presently support symbols, when you try to execute the **sym** command, a message will be displayed indicating that symbols are not supported on your emulator.

If your emulator firmware is less than version A.02.00, you will not be able to use the **sym** command because your emulator will not support symbols. To verify the version number of your emulator firmware, execute "ver" at the Terminal Interface prompt.

Even if your emulator firmware version is A.02.00 or greater, your HP 64700-Series emulator may not necessarily support symbols.

Syntax A symbol file is an ASCII text file. The format of this file is represented by:



<WHITESPACE>	This is one or more <SP> (space) or <HT> (horizontal tab) characters or a combination of these characters.
<RETURN>	This is a <LF> (line feed) or <CR><LF> (carriage return, line feed pair); a <CR> (carriage return) alone is not recognized.
<ADDRESS>	This is a valid address specification for the emulator being used.
<MODULE>	This defines a module name.
<LOCAL SYMBOL>	This is a local symbol reference. A local symbol definition line must include, or follow, a module name, or an error will occur when loading the file.
<GLOBAL SYMBOL>	This is a global symbol reference.
<QUALIFIER>	This allows you to specify label hierarchies. Its use is dependent on the implementation.
:	This is the literal colon (":").
.	This is the literal period (".")
#	This is the literal pound sign ("#").

Examples

The examples presented are for the 8051 family emulators. Other emulators will use a different address format. Refer to your *Emulator Terminal Interface User's Guide* for specific address format definitions.

Defining Local Symbols

Local symbols must include, or be preceded by, a module name reference. Therefore, the files

```
#
:main 0@p
GetAttrib:
Buffer 100@p
Pointer 120@p
#
```

and

```
#
:main 0@p
GetAttrib:Buffer 100@p
GetAttrib:Pointer 120@p
#
```

will produce the same result when loaded.

After loading either symbol file, enter:

```
M> sym
```

You will see:

```
sym main=00000@p
sym GetAttrib:Buffer=00100@p
sym GetAttrib:Pointer=00120@p
```

Naming Array Elements

You may wish to load symbols that name elements of an array to make referring to the array elements more explicit. If your array has four elements, each element is 10h bytes long, and begins at 2000h, the symbol file would contain the following:

```
#
ARRAY:
E1=2000@d
E2=2010@d
E3=2020@d
E4=2030@d
#
```

After loading the symbol file, enter:

```
M> sym
```

You will see, at least in part:

```
sym ARRAY:E1=2000@d
sym ARRAY:E2=2010@d
sym ARRAY:E3=2020@d
sym ARRAY:E3=2030@d
```

If you no longer need the references to ARRAY elements, you can remove the symbols with the command:

```
M> sym -dl ARRAY
```

Loading a Symbol File

A symbol file can be loaded from a host system using the **load** command with an HP 64700 transparent configuration. The procedure is similar to that used to load a program. The following commands will load the symbol file named "symbolfile" into the emulator:

```
M> load -So <RETURN>
$ cat symbolfile <ESC>G
```

You can also load the file using the "string" (-s) option with the **load** command. After you have logged on to the host system, the following command will also load the symbol file:

```
M> load -Sos "cat symbolfile"
```

8051 Sample Program

The HP 64700-Series 8051 Emulator supports the use of symbols. If you are using an 8051 Emulator, you can use the following program to become familiar with using symbols.

```
"8051"
DPS             DATA    86H
                GLB     Msgs,Init,Cmd_Input
                CSEG
                COMN
Msgs
Msg_A           DB      "Command A entered "
Msg_B           DB      "Entered B command "
Msg_I           DB      "Invalid Command "
End_Msgs

                PROG
*****************************************************
* The following instructions initialize segment
* registers and set up the stack pointer.
*****************************************************
Init            MOV     SP,#Stk
Read_Cmd        MOV     DPTR,#Cmd_Input
*****************************************************
* Clear previous command.
*****************************************************
                MOV     A,#0
                MOVX    @DPTR,A
*****************************************************
* Read command input byte.  If no command has been
* entered, continue to scan for command input.
*****************************************************
Scan            MOVX    A,@DPTR
                JZ      Scan
*****************************************************
* A command has been entered.  Check if it is
* command A, command B, or invalid.
*****************************************************
                CJNE    A,#41H,Cmd_B
*****************************************************
* Command A is entered.  Set up registers R2 and
* DPTR0 with the parameters expected by the
* "Write_Msg" routine.  Call the routine.  After
* return, go back and scan for next command.
*****************************************************
                MOV     R2,#Msg_B-Msg_A
                MOV     DPTR,#Msg_A
                SJMP    Write_Msg
Cmd_B           CJNE    A,#42H,Cmd_I
*****************************************************
* Command B is entered.
*****************************************************
                MOV     R2,#Msg_I-Msg_B
                MOV     DPTR,#Msg_B
                SJMP    Write_Msg
*****************************************************
* An invalid command is entered.
*****************************************************
Cmd_I           MOV     R2,#End_Msgs-Msg_I
                MOV     DPTR,#Msg_I
```

```
*****************************************************
* The "Write_Msg" routine writes a message to the
* destination.  The parameter passed in register
* R2 = the number of bytes in the message.  The
* parameter passed in register DPTR0 = the location
* of the message.
*****************************************************
Write_Msg       MOV     A,R2
                XRL     A,#0FFH
                ADD     A,#21H
                MOV     R3,A
                INC     DPS
                MOV     DPTR,#Msg_Dest
Again           INC     DPS
                MOVX    A,@DPTR
                INC     DPTR
                INC     DPS
                MOVX    @DPTR,A
                INC     DPTR
                DJNZ    R2,Again
*****************************************************
* The rest of the destination area is filled
* with zeros.
*****************************************************
                MOV     A,#0
Fill_Dest       MOVX    @DPTR,A
                INC     DPTR
                DJNZ    R3,Fill_Dest
                INC     DPS
                SJMP    Read_Cmd

                XSEG
*****************************************************
* Command input byte.
*****************************************************
Cmd_Input       DS      1
*****************************************************
* Destination of the command messages.
*****************************************************
Msg_Dest        DS      0FDH
Stk             DS      1       ; Stack area.
                END     Init
```

Notes

Binary/Hexadecimal Trace List Format

The **tl** command supports two options, **-b** (binary) and **-x** (hexadecimal) which allow you to dump the trace list to your host for post processing. This is, in fact, the only way you can obtain timing trace information from the optional external analyzer, as the Terminal Interface does not currently provide support for translating timing data to ASCII characters.

Transfer Protocol

When you request a binary trace list dump from the HP 64700 Emulator (**-b** option), the emulator sends the data using the HP 64000 **transfer** protocol. You must use an 8-bit communications channel to successfully transfer the data (HP 64700 and the host device must both be configured to send and receive 8 bits).

The hexadecimal trace list dump (**-x** option) also uses the HP 64000 **transfer** protocol, but does not require an 8-bit communications channel. However, twice as many characters will be transmitted as would be in the binary format.

Trace List Records

Six primary trace list records may be transferred. These are:

- No Trigger Record
- Empty Trace Record
- New State Data Record

- More State Data Record
- New Timing Data Record
- More Timing Data Record

Each record has at least one byte. The first byte identifies the record type.

Other fields in the record, containing one or more bytes of information, provide additional information about the trace.

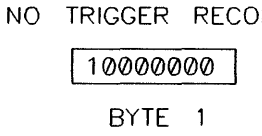
The Data Records contain secondary record structures which hold the actual trace information. For the State Data Records, the secondary record is the Trace State record; for the Timing Data Records, the secondary record is the Trace Sample record.

Each record structure is accompanied by a diagram. Note that line breaks in the diagram are not EOL characters in the record.

No Trigger Record

record type = 10000000

One byte indicating that the trigger condition of the current trace is not in memory. Trace data cannot be displayed until the trigger condition occurs and is placed in trace memory or until the trace is halted. Therefore, this is the only record that will be sent when the trace list is requested, since no others are available.



Empty Trace Record

record type = 01000000

One byte indicating that the most recent trace was halted before any states were stored. Therefore, this will be the only record sent.

EMPTY TRACE RECORD

01000000

BYTE 1

New State Data Record

record type = 00000LH1

One byte indicating that this is the first trace list data displayed for the current or most recent trace.

If L=1, this is the only record being sent. Otherwise, one or more More Data Records follow.

If H=1, this record contains the highest numbered state this trace can have. Therefore, this is the end of the trace list. If the state count for this record is zero, the highest numbered state can be computed by subtracting 1 from the start state.

state count

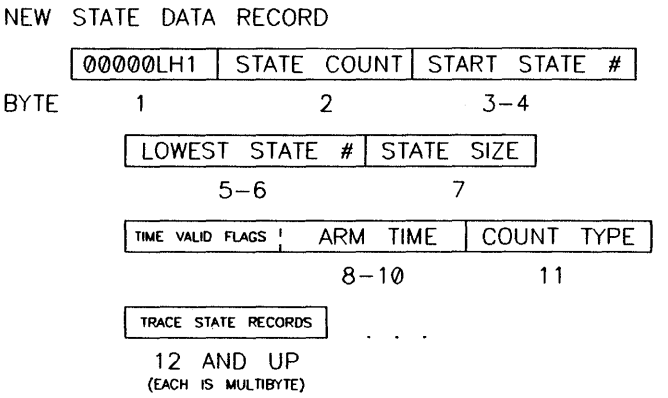
One byte indicating how many trace states are contained in this record. This will be zero (0) if none of the requested states exist.

start state

Two bytes containing the starting state number (in the range -1024 through 1023), most significant byte first.

lowest state

Two bytes containing the lowest state number in the entire trace list, MSB first. Note that if the trace is halted after this record is sent, lower-numbered states may be valid.



state size

One byte indicating how many bytes of trace data will be in each trace state. This does not include the store cause or count data bytes.

arm time

Three bytes containing the time from arm to trigger, MSB first. The lower 20 bits contain the absolute value of the actual time, in 40 ns units.

Note 

The time alignment between HP 64700-Series emulators has a large margin of error (+/- 100 ns) due to delay variances in the trigger paths.

The correlation between the arm time counter value and the value displayed on screen should be as follows:

```
count       time
-----       ------------------------------------------------------
0000h       Arm occured an unknown amount of time after the trigger
0001h       Arm occured an unknown amount of time after the trigger
0002h      -40 ns - Arm input actually came after trigger was sampled but
                    still caused arm state to occur before trigger
                    internal to the elan chip.
0003h        0 ns
0004h       40 ns
0005h       80 ns
  .          .
  .          .
  .          .
FFFFh       2.621280 ms  This is now the maximum arm to trigger interval
                         that can be displayed.
```

The highest 4 bits contain status flags as follows:

high nibble = XVS0

If X = 1, the arm time is invalid, either because the arm signal was ignored (e.g., "tarm always"), or because the state analyzer clock speed was fast or very fast (e.g., "xtck -s F"). The 20 bits of time value will be 0.

If V = 1, the arm counter overflowed (if S = 0) or underflowed (if S = 1). For overflow, the 20 bits of time value contain the maximum time value, (1 ^ 20)-4, representing 41.94288 ms. For underflow, the S flag is set (see below), and the 20 bits of time value contain the absolute value of the minimum count, -1, representing -40 ns.

If S = 1, the arm time is negative. The 20 bits of time value contain the absolute value of the actual count.

count type

One ASCII character indicating the type of count data contained in each trace state.

"T" indicates each trace state contains a time count.

"S" indicates each trace state contains a state count.

"N" indicates that no count data is available.

first trace state..last trace state

Each of these records is in the trace state format described below. Each record is n bytes in length; n is the state size value (described above) plus one byte indicating the reason for storage of this state and an optional two bytes with count data information.

More State Data Record

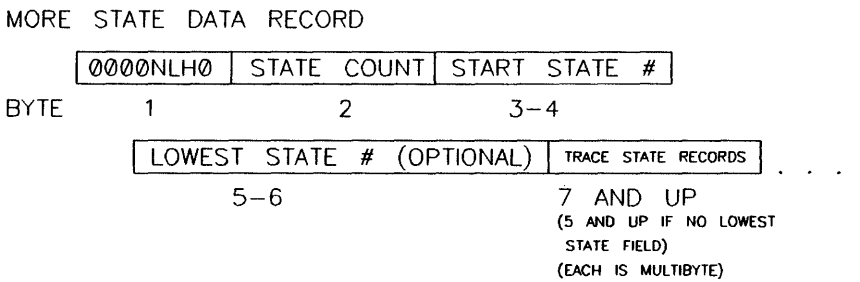
record type = 0000NLH0

One byte indicating this is more data from the same trace as the most recent New State Data Record.

If L=1, this is the last record sent. Otherwise additional More Data Records follow.

If H=1, this record contains the highest numbered state in the trace; this is the end of the trace list. If the state count for this record is zero (0), the highest numbered state can be computed by subtracting one (1) from the start state.

If N=1, this record contains a new lowest state. The starting state number can change if the trace is halted; if it changes, it will always become more negative. It can change a maximum of one time for a given trace list. N=1 will never occur unless L=1.



state count

One byte indicating the number of trace states contained in this record. This will be zero (0) if none of the requested states exist.

start state

Two bytes containing the starting state number (in the range -1024..1023), most significant byte (MSB) first.

lowest state

Optional two bytes containing the lowest state number in the entire trace list, most significant byte (MSB) first. These bytes are only present if the record type has N=1.

first trace state..last trace state

Each of these records is in the trace state format described below. Each record is a variable number of bytes in length. The length is the state size value (described above) plus one byte indicating the reason for storage of this state and an optional two bytes with count data information.

Trace State Record

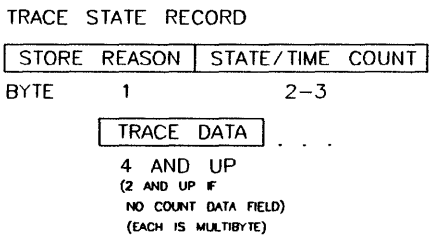
state type

One ASCII character indicating the reason this state was stored.

"Q" indicates this state satisfied a sequence branch qualifier (defined by **tif** or **telif**).

"S" indicates this state satisfied the store qualifier (defined by **tsto**).

"P" indicates this state satisfied the prestore qualifier. The count data field bytes below will be omitted for this state. Prestore states are marked as such only if a state or time count was specified for the trace (defined by **tcq**).



count data

Optional two bytes containing the state or time count for this state. The count value is relative to the previous non-prestore state. These bytes are omitted if the count type field in the New State Data Record was "N", or if this state is a prestore state (state type field in this record is "P"). The count data is encoded as follows (first byte is on the left):

eeeeemmm mmmmmmmm

e represents 5 bits of exponent.

m represents 11 bits of mantissa.

The value represented is (m*(2^e)) + (2^(11+e)) - (2^11)

Time counts are in 40 nanosecond units.

trace data

Trace data for this state, most significant byte (MSB) first. The length of this trace data is given by the state size field in the New State Data Record. Data from the optional external analyzer (provided only if the analyzer is present) will be in the most significant 16 bits (two bytes).

New Timing Data Record

record type = 00100LH1

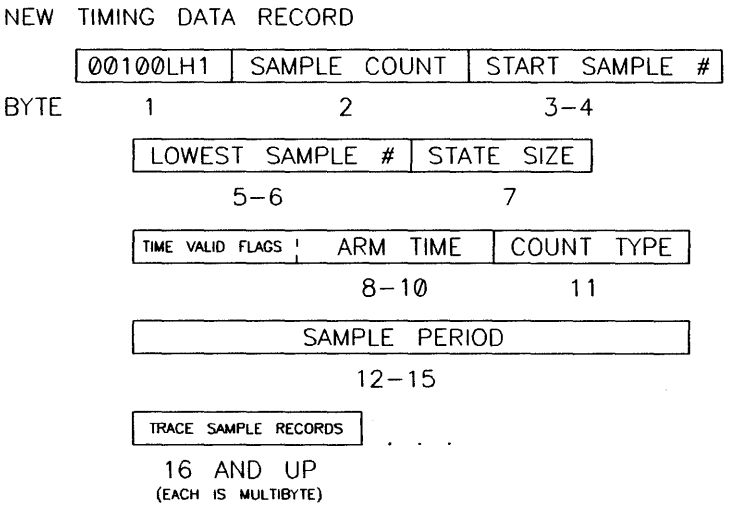
One byte indicating this is the first trace list data displayed for the current or most recent trace.

If L=1, this is the only record sent. Otherwise, one (1) or more More Timing Data Records follow.

If H=1, this record contains the highest numbered sample in the trace; this is the end of the trace list. If the sample count field for this record is zero (0), the highest numbered sample can be computed by subtracting one (1) from the start sample field.

sample count

One byte indicating how many trace samples are contained in this record. This will be zero if no samples are present.



start sample

Two bytes containing the starting sample number (-1024..1023), most significant byte (MSB) first.

lowest sample

Two bytes containing the lowest sample number in the entire trace list, MSB first. Note that if the trace is halted after this record is sent, lower-numbered samples may become valid.

state size

One byte indicating the number of bytes of trace data in each trace sample. Note the relationship to the count type field.

arm time

Three bytes containing the time from arm to trigger, MSB first. The lower 20 bits contain the absolute value of the actual time, in 40 ns units.

Note 

The time alignment between HP 64700-Series emulators has a large margin of error (+/- 100 ns) due to delay variances in the trigger paths.

The correlation between the arm time counter value and the value displayed on screen should be as follows:

```
count       time
-----       ------------------------------------------------------
0000h       Arm occured an unknown amount of time after the trigger
0001h       Arm occured an unknown amount of time after the trigger
0002h      -40 ns - Arm input actually came after trigger was sampled but
                    still caused arm state to occur before trigger
                    internal to the elan chip.
0003h       0 ns
0004h       40 ns
0005h       80 ns
  .          .
  .          .
  .          .
FFFFh       2.621280 ms  This is now the maximum arm to trigger interval
                         that can be displayed.
```

The highest 4 bits contain status flags as follows:

high nibble = XVS0

If X = 1, the arm time is invalid, either because the arm signal was ignored (e.g., "tarm always"), or because the state analyzer clock speed was fast or very fast (e.g., "xtck -s F"). The 20 bits of time value will be 0.

If V = 1, the arm counter overflowed (if S = 0) or underflowed (if S = 1). For overflow, the 20 bits of time value contain the maximum time value, (1 ^ 20)-4, representing 41.94288 ms. For underflow, the S flag is set (see below), and the 20 bits of time value contain the absolute value of the minimum count, -1, representing -40 ns.

If S = 1, the arm time is negative. The 20 bits of time value contain the absolute value of the actual count.

count type

One ASCII character indicating the type of count data contained in each Trace Sample record.

"T" indicates the timing analyzer was set to transitional mode. Each Trace Sample record contains a six byte field which contains the delta time (in nanoseconds) since the last transition. A two-byte field containing the trace data taken at the delta time interval is also in the Trace Sample record.

"S" indicates the timing analyzer was set to standard mode. Each Trace Sample record contains only the two bytes of trace data.

"G" indicates the timing analyzer was set to glitch mode. Each trace sample consists of a two-byte trace data field and a two-byte glitch data field.

sample period

Four bytes containing the number of nanoseconds (ns) between samples.

first trace sample..last trace sample

Trace Sample records of the size defined in the sample size field (note relationship to the count type field).

More Timing Data Record

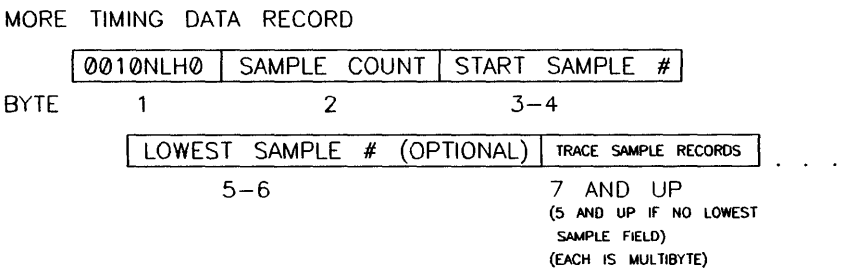
record type = 0010NLH0

One byte indicating this is more data from the same trace as the most recent New Timing Data Record.

If L=1, this is the last record sent. Otherwise, additional More Timing Data Records follow.

If H=1, this record contains the highest-numbered sample in the trace; this is the end of the trace list. If the sample count field for this record is zero (0), the highest numbered sample can be computed by subtracting one (1) from the start sample field.

If N=1, this record contains a new lowest sample. The starting sample number can change if the trace is halted; if it changes, it will always become more negative. It can only change once for a given trace list. N=1 will only occur if L=1.



sample count

One byte indicating the number of Trace Sample records in this record. This will be zero (0) if no Trace Samples are present (the analyzer did not find the requested data in the last trace.)

start sample

Two bytes containing the starting sample number (in the range -1024..1023), most significant byte (MSB) first.

lowest sample

Optional two bytes containing the lowest sample number in the entire trace list, most significant byte (MSB) first. These two bytes are present only if the record type has N=1.

first trace sample..last trace sample

Trace Sample records of the size defined in the sample size field (note relationship to the count type field).

Trace Sample Records

Trace Sample records are variant records which are components of the New Timing Data Record and More Timing Data Record. The structure of the Trace Sample Record depends on the count type field in the Timing Data Records.

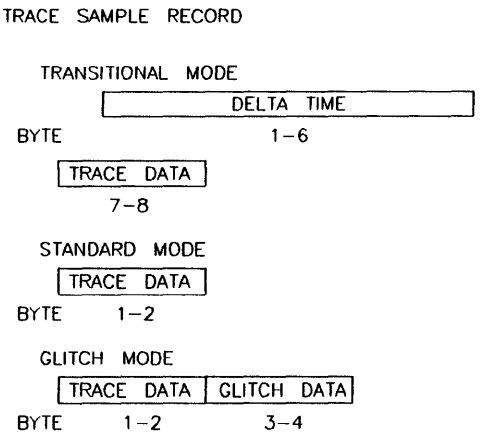
Transitional Mode (count type = "T")

delta time

Six bytes of data defining the delta time (elapsed time) since the last transition, in nanoseconds (ns).

trace data

Two bytes of trace data sampled at the delta time value given.



Standard Mode (count type = "S")

trace data

Two bytes of trace data sampled at the standard sampling period (see the **xtsp** command).

Glitch Mode (count type = "G")

trace data

Two bytes of trace data sampled at the standard sampling period (see the **xtsp** command).

glitch

Two bytes indicating the occurrences of glitches on any channel.

C

Error Messages

This appendix contains descriptions of error messages that can occur while using the Terminal Interface. The error messages are listed in numerical order, and each description includes the cause of the error and the action you should take to remedy the situation.

Error messages described in this appendix are "generic"; that is, they can occur in any of the HP 64700-Series emulators. Errors specific to a particular emulator are described in the *Emulator User's Guide.*

The HP 64700-Series emulators can return messages to the display only when they are prompted to do so. Situations may occur where an error is generated as the result of some command, but the error message is not displayed until the next command (or a carriage return) is entered.

A maximum number of 8 error messages can be displayed at one time. If more than 8 errors are generated, only the last 8 are displayed.

Emulator Error Messages

The following messages are used by most, but not all, of the HP 64700-Series Emulators. Some emulators may supplement or replace these with messages of their own.

Message 0 : Software breakpoints not supported

Cause

You attempted to enable software breakpoints with **bc -e bp** on an emulator whose processor does not support a software breakpoint instruction.

Action

Do not attempt to use software breakpoints. Instead, set the analyzer to drive the **trig1** or **trig2** line upon finding trigger, specify the trigger state as the address you wish to break on, trace, then run the emulator. The emulator will break to monitor when the analyzer finds the trigger state.

Message 1 : I/O port access not supported

Cause

You attempted to use the **io** command for an emulator whose processor does not support separate I/O (such as the 68000).

Action

Use the **m** command to modify I/O ports on these emulators.

Message 20 : Attempt to change foreground monitor map term

Cause

The **cf mon=fg** command that sets up use of a foreground monitor also maps a memory range for the monitor's use. You attempted to alter that term using the **map** command.

Action

Try using another memory range for the new map term. If you need to have the range used by the foreground monitor, then switch to a background monitor, delete the old foreground monitor map term, and add the new term. Now you can return to using a

foreground monitor; remember you will need to reload the monitor code.

Message 40 : Restricted to real time runs

Cause

The **cf rrt=en** option is set (restrict to real time runs) and you have entered a command which requires a temporary break to the monitor for processing (such as a request to display target system memory locations).

Action

Break to the monitor using the **b** command, then execute the desired command. If your target system depends on continuous bus cycles to avoid damage, then power it down before breaking to the monitor.

Message 61 : Emulator is in the reset state

Cause

This message is displayed if you request an operation that requires entry into the emulation monitor, such as display of target system memory locations.

Action

If the prompt is R>, indicating an emulation system reset, break to the monitor using the **b** command, then retry the command. Otherwise, release the target system reset, then retry the command.

Message 80 : Stack pointer is odd

Cause

You have attempted to modify the stack pointer to an odd value for a processor that expects the stack to be aligned on a word boundary (such as the 68000).

Action

Modify the stack pointer to an even value.

Message 81 : Stack is in guarded memory

Cause

Your stack pointer pointed to a location in memory mapped as guarded; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action

Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or with the **cf** command options, if available) to point to a location in RAM.

Message 82 : Stack is in target ROM

Cause

Your stack pointer pointed to a location in memory mapped as target ROM; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action

Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or with the **cf** command options, if available) to point to a location in RAM.

Message 83 : Stack is in emulation ROM

Cause

Your stack pointer pointed to a location in memory mapped as emulation ROM; you then attempted to run or step the emulation processor. The emulator was unable to access the stack to complete the transition from the monitor to the user program or vice versa.

Action

Either remap memory so the stack pointer points to a location in RAM, or change the stack pointer value (either with your program or with the **cf** command options, if available) to point to a location in RAM.

Message 84 : Program counter is odd

Cause

You attempted to modify the program counter to an odd value using the **reg** command on a processor which expects even alignment of opcodes.

Action

Modify the program counter only to even numbered values.

Message 102 : Monitor failure; no clock input

Cause

The monitor is unable to run because no emulation processor clock is available.

Action

If running out of circuit, choose configuration option **cf clk=int**; if running in-circuit, choose configuration option **cf clk=ext** and

make sure a clock meeting the microprocessor's specifications is input to the clock pin of the target system probe.

Message 103 : Monitor failure; no processor cycles

Cause

The monitor is unable to run since the processor is not running. The monitor is unable to determine the cause of the failure.

Action

If running in-circuit, troubleshoot the target system. If running out of circuit, reinitialize the emulator and try the procedure again.

Message 104 : Monitor failure; bus grant

Cause

The monitor is unable to run. The emulation processor is not running because it has granted the bus to another device.

Action

Wait until the processor has regained bus control, then retry the operation.

Message 105 : Monitor failure; halted

Cause

The monitor is unable to run because the processor is halted (due to an external halt line or a halt instruction).

Action

Release the external halt and retry the operation. If the processor halted due to a halt instruction, try the **rst** command, then retry the operation.

Message 106 : Monitor failure; wait state

Cause

The monitor is unable to run because the processor is in a continuous wait state.

Action

A continuous wait state may indicate target system problems. Troubleshoot the wait line. If you were running out of circuit, try initializing the emulator with **init**, then retry the procedure.

Message 107 : Monitor failure; bus error

Cause

The monitor is unable to run because the processor has encountered a bus fault (such as the 68000 /BERR line).

Action

Release the /BERR line and determine why it was activated.

General Emulator and System Error/Status Messages

Message 201 : Out of system memory

Cause

Macros and equates that you have defined have used all of the available system memory.

Action

Delete some of the existing macros (**mac -d <NAME>**) and equates (**equ -d <NAME>**), which will free additional memory.

Message 204 : FATAL SYSTEM SOFTWARE ERROR

205 : FATAL SYSTEM SOFTWARE ERROR

208 : FATAL SYSTEM SOFTWARE ERROR

Cause

The system has encountered an error from which it cannot recover.

Action

Write down the sequence of commands which caused the error. Cycle power on the emulator and reenter the commands. If the error repeats, call your local HP Sales and Service office for assistance.

Message 206 : Incompatible compatibility table entry.

Cause

The emulation firmware (ROM) is not compatible with the analysis or system firmware in your HP 64700 emulation system.

Action

The ROMs in your emulator must be compatible with each other for your emulation system to work correctly. Refer to the HP 64700-Series Emulators Firmware/Software Compatibility Note supplied with your HP 64700-Series documentation to determine the current ROM kit number for your emulator. All emulation, analysis, and system ROMs in the HP 64700 must be installed from the most current firmware kit for that emulator. For more information contact your Hewlett-Packard Representative.

Message 300 : Invalid option or operand

305 : Invalid option or operand: %s

Cause

You have specified incorrect option(s) to a command. %s, if printed, indicates the incorrect option(s).

Action

Reenter the command with the correct syntax. Refer to the syntax pages in this example for information.

Message 307 : Invalid expression: %s

Cause

You have entered an expression with incorrect syntax; therefore, it cannot be evaluated. %s is the bad expression.

Action

Reenter the expression, following the syntax rules for that type of expression. Refer to the command syntax pages to determine the expression type; then refer to the expression syntax pages to determine the correct syntax for that type.

Message 308 : Invalid number of arguments

Cause

You have either entered too many options to a command or an insufficient number of options.

Action

Re-enter the command with correct syntax. Refer to the command syntax pages in this manual for information.

Message 310 : Invalid address: %s

Cause

You specified an invalid address value as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number (even zero (0)).

Action

Re-enter the command and the address specification. See the <ADDRESS> syntax pages in the *Emulator User's Guide* and the <EXPRESSION> syntax pages in this manual for information on address specifications.

Message 311 : Invalid address range: %s

Cause

You specified an invalid address range as an argument to a command (other than an analyzer command). For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number, or the upper boundary of the range you specified is less than the lower boundary.

Action

Re-enter the command and the address specification. See the <ADDRESS> syntax pages in the *Emulator User's Guide* and the <EXPRESSION> syntax pages in this manual for information on address specifications. Also, make sure that the upper boundary specification is greater than the lower boundary specification (the lower boundary must always precede the upper boundary on the command line).

Message 312 : Ambiguous address: %s

Cause

Certain emulators support segmentation or function code information in addressing. The emulator is unable to determine which of two or more address ranges you are referring to, based upon the information you entered.

Action

Re-enter the command and fully specify the address, including segmentation or function code information.

Message 313 : Missing option or operand

Cause

You have omitted a required option to the command.

Action

Re-enter the command with the correct syntax. Refer to the command syntax pages in this manual for further information on required syntax.

Message 314 : Option conflict: %s

Cause

You have entered a command with two options which cannot be used together. For example, you might have entered **tl -bx**; you cannot ask for both a binary and hexadecimal trace list dump.

Action

Reenter the command, specifying only non-conflicting options. Refer to the syntax pages for the command in this manual to determine which options may be used together.

Message 315 : Invalid count: %s

Cause

This error occurs when the emulation system expects a certain number (of arguments, for example), but you specify a different number.

Action

Enter the number the system expects to receive.

Message 316 : Invalid range expression: %s

Cause

In the **tl** command, you specified an illegal range. For example, you might have specified **tl -10..a**.

Action

Use only legitimate range numbers in the **tl** command (-1024..1023); the second range value must be greater than the first.

Message 317 : Range out of bounds: %s

Cause

In the **tl** command, you specified a range number which was greater than the number of states available in the analyzer. For example, you might have specified **tl -2048..2048**; the analyzer only has 1024 states.

Action

Specify range numbers between -1024 and 1023.

Message 318 : Count out of bounds: %s

Cause

You specified an occurrence count less than 1 or greater than 65 535 for **tg** or **tif**. For example, you might have entered **tif 1 any 2 69234**.

Action

Re-enter the command, specifying a count value from 1 to 65535. For example: **tif 1 any 2 65535**.

Message 319 : Invalid base: %s.

Cause

This error occurs if you have specified an invalid base in the **tf** or **xtf** commands.

Action

Enter the **help tf** or **help xtf** command to view the valid base options.

Message 320 : Invalid label: %s

Cause

You tried to define a label with characters other than letters, digits, or underscores.

Action

Re-enter the **tlb** command with a label consisting only of letters, digits, or underscores.

Message 321 : Label not defined: %s

Cause

You entered an analyzer expression in which the label was not present in the analyzer label list. For example, if the label list includes **addr**, **data**, and **stat**, you might have entered something such as **tg lowerdata=24t**. This error also occurs if you try to delete a label that does not exist.

Action

You can re-enter the command, using one of the previously defined labels and adjust the expression as necessary to accommodate the fit of that label to the analyzer input lines. Or, you can define a new label using the **tlb** command, then re-enter the analyzer command using the newly defined label.

Message 400 : Record checksum failure

Cause

During a **transfer** operation, the checksum specified in a file did not agree with that calculated by the HP 64700.

Action

Retry the **transfer** operation. If the failure is repeated, make sure that both your host and the HP 64700 data communications parameters are configured correctly.

Message 401 : Records expected: %s; records received: %s

Cause

The HP 64700 received a different number of records than it expected to receive during a **transfer** operation.

Action

Retry the **transfer**. If the failure is repeated, make sure that the data communications parameters are set correctly on the host and on the HP 64700. Refer to the *Hardware Installation and Configuration* manual for details.

Message 410 : File transfer aborted

Cause

A **transfer** operation was aborted due to a break received, most likely a <CTRL> c from the keyboard.

Action

If you typed <CTRL> c, you probably did so because you thought the transfer was about to fail. Retry the transfer, making sure to use the correct command options. If you are unsuccessful, make sure that the data communications parameters are set correctly on the host and on the HP 64700, then retry the operation.

Message 411 : Severe error detected, file transfer failed

Cause

An unrecoverable error occurred during a **transfer** operation.

Action

Retry the transfer. If it fails again, make sure that the data communications parameters are set correctly on the host and on the HP 64700. Also make sure that you are using the correct command options, both on the HP 64700 and on the host.

Message 412 : Retry limit exceeded, transfer failed

Cause

The limit for repeated attempts to send a record during a **transfer** operation was exceeded, therefore the transfer was aborted.

Action

Retry the transfer. Make sure you are using the correct command options for both the host and the HP 64700. The data communications parameters need to be set correctly for both devices. Also, if you are in a remote location from the host, it is possible that line noise may cause the failure.

Message 413 : Transfer failed to start.

Cause

Communication link or transfer protocal incorrect.

Action

Check link and transfer options.

Message 415 : Timeout, receiver failed to respond.

Cause

Communication link or transfer protocal incorrect.

Action

Check link and transfer options.

Message 420 : Unknown mode: %s.

Cause

This error occurs when you have specified an unknown option in the **stty** command.

Action

Enter the **help stty** command to view the valid options.

Message 425 : Load option conflict: %s and option: %s.

Cause

Two or more options in the load command cannot be used together.

Action

Enter the **help load** command to view the options that cannot be used together.

Message 520 : Equate not defined: %s

Cause

You tried to delete an equate that did not exist in the equate table. For example suppose the equates **a=1** and **b=2** were in the equate table. If you typed **equ -d c**, you would receive the above error message.

Action

Use **equ** to display the list of named equates before deleting equates.

Message 600 : Adjust PC failed during break.

Cause

System failure or target condition.

Action

Run performance verification (**pv** command), and check target system.

Message 602 : Break failed

Cause

The **b** command was unable to break the emulator to the monitor.

Action

Determine why the break failed, then correct the condition and retry the command. See message 608.

Message 603 : Read PC failed during break.

Cause

System failure or target condition.

Action

Try again.

Message 604 : Disable breakpoint failed: %s.

Cause

System failure or target condition.

Action

Run performance verification (**pv** command), and check target system.

Message 605 : Undefined software breakpoint: %s

Cause

The emulator has encountered a software breakpoint in your program that was not inserted with the **bp** command.

Action

If your processor allows different software breakpoint instructions, either modify the ones you inserted in your code, or modify the ones inserted by **bp** using your emulator's configuration options (**cf** command). If only one instruction is available, remove those inserted in your code before assembly and link, then reinsert them using the **bp** command.

Message 606 : Unable to run after CMB break.

Cause

System failure or target condition.

Action

Run performance verification (**pv** command), and check target system.

Message 608 : Unable to break

Cause

This message is displayed if the emulator is unable to break to the monitor because the emulation processor is reset, halted, or is otherwise disabled.

Action

First, look at the emulation prompt and other status messages displayed to determine why the processor is stopped. If reset by the emulation controller, use the **b** command to break to the monitor. If reset by the emulation system, release that reset. If halted, try **rst -m** to get to the monitor. If there is a bus grant, wait for the requesting device to release the bus before retrying the command. If there is no clock input, perhaps your target system is faulty or you have configured the clock wrong with **cf clk**. It's also possible that you have configured the emulator to restrict to real time runs, which will prohibit temporary breaks to the monitor. Refer to Appendix D of this manual for a list of emulation prompts and their meanings.

Message 610 : Unable to run.

Cause

System failure or target condition.

Action

Run performance verification (**pv** command), and check target system.

Message 611 : Break caused by CMB not ready

Cause

This status message is printed during coordinated measurements if the CMB READY line goes false. The emulator breaks to the monitor. When CMB READY is false, it indicates that one or more of the instruments participating in the measurement is running in the monitor.

Action

None, information only.

Message 612 : Write to ROM break

Cause

This status message will be printed if you have set **bc -e rom** and the emulation processor attempted a write to a memory location mapped as ROM.

Action

None (except troubleshooting your program!)

Message 613 : Analyzer Break.

Cause

Status message.

Action

None.

Message 614 : Guarded memory access break

Cause

This message is displayed if the emulation processor attempts to read or write memory mapped as guarded.

Action

Troubleshoot your program; or, you may have mapped memory incorrectly.

Message 615 : Software breakpoint: %s

Cause

This status message will be displayed if a software breakpoint entered with **bp** and enabled with **bc -e bp** is encountered during a program run. The emulator is broken to the monitor. The string %s indicates the address where the breakpoint was encountered.

Action

None.

Message 616 : BNC trigger break

Cause

This status message will be displayed if you have set **bc -e bnct** and the BNC trigger line is activated during a program run. The emulator is broken to the monitor.

Action

None.

Message 617 : CMB trigger break

Cause

This status message will be displayed if you have set **bc -e cmbt** and the CMB trigger line is activated during a program run. The emulator is broken to the monitor.

Action

None.

Message 618 : trig1 break

Cause

This status message will be displayed if you have set the analyzer to drive **trig1** upon finding the trigger, **bc -e trig1** is set, and the analyzer has found the trigger condition while tracing a program run. The emulator is broken to the monitor.

Action

None.

Message 619 : trig2 break

Cause

This status message will be displayed if you have set the analyzer to drive **trig2** upon finding the trigger, **bc -e trig2** is set, and the analyzer has found the trigger condition while tracing a program run. The emulator is broken to the monitor.

Action

None.

Message 620 : Unexpected software breakpoint

Cause

If you have enabled software breakpoints with **bc -e bp**, this message is displayed if a software breakpoint instruction is encountered in your program that was not inserted by **bp** and is therefore not in the breakpoint table.

Action

If your processor allows different software breakpoint instructions, either modify the ones you inserted in your code, or modify the ones inserted by **bp** using your emulator's configuration options (**cf** command). If only one instruction is available, remove those inserted in your code before assembly and link, then reinsert them using the **bp** command.

Message 621 : Unexpected step break.

Cause

System failure.

Action

Run performance verification (**pv** command).

Message 622 : %s.

Cause

Monitor specific message.

Action

Refer to your *Emulator User's Guide* for these message descriptions.

Message 623 : CMB execute break.

Cause

This message occurs when coordinated measurements are enabled and an EXECUTE pulse causes the emulator to run; the emulator must break before running.

Action

This is a status message; no action is required.

Message 624 : Configuration aborted.

Cause

Occurs when a <CTRL> c is entered during **cf** display command.

Action

None.

Message 625 : Invalid configuration value: %s

Cause

You have entered a configuration option incorrectly, such as typing **cf clk=ex** instead of **cf clk=ext**.

Action

Re-enter the configuration command, specifying only the correct options. Refer to the *Emulator User's Guide* for a description of the configuration options for your emulator.

Message 626 : Configuration failed; setting unknown: %s=%s.

Cause

Target condition or system failure.

Action

Check target system, and run performance verification (**pv** command).

Message 627 : Invalid configuration item: %s

Cause

You specified a non-existent configuration item in the **cf** command. For example, if your emulator only supports a background monitor, you would see this message if you tried to enter **cf mon=fg** since there is no **mon** configuration item for your emulator.

Action

Re-enter the command, specifying only configuration items that are supported by your emulator. Refer to the <CONFIG_ITEMS> syntax pages in your *Emulator User's Guide* for further information.

Message 630 : Register access aborted.

Cause

Occurs when a <CTRL> c is entered during register display.

Action

None.

Message 631 : Unable to read registers in class: %s

Cause

The emulator was unable to read the registers you requested.

Action

To resolve this, you must look at the other status messages displayed. Most likely, the emulator was unable to break to the monitor to perform the register read. See message 608.

Message 632 : Unable to modify register: %s=%s

Cause

The emulator was unable to modify the register you requested.

Action

To resolve this, you must look at the other status messages displayed. It's likely that emulator was unable to break to the monitor to perform the register modification. See message 608.

Message 634 : Display register failed: %s

Cause

The emulator was unable to display the register you requested.

Action

To resolve this, you must look at the other status messages displayed. It's likely that emulator was unable to break to the monitor to perform the register display. See message 608.

Message 636 : Register not writable: %s.

Cause

This error occurs when you attempt to modify a read only register.

Action

If this error occurs, you cannot modify the contents of the register with the **reg** command.

Message 637 : Register class cannot be modified: %s

Cause

You tried to modify a register class instead of an individual register.

Action

You can only modify individual registers. Refer to the <REGISTERS> syntax pages in the *Emulator User's Guide* for a list of register names.

Message 640 : Unable to reset.

Cause

Target condition or system failure.

Action

Check target system, and run performance verification (**pv** command).

Message 641 : Unable to reset into monitor.

Cause

You have entered a **rst -m** command and the emulator is unable to break into the monitor.

Action

Reload monitor (**rst** for background).

Message 650 : Unable to configure break on write to ROM

Cause

The emulator controller is unable to execute the **bc -e rom** command correctly, possibly because the emulator was left in an unknown state or because of a hardware failure.

Action

Initialize the emulator or cycle power. Then reenter the command. If the same failure occurs, call your HP sales and service office.

Message 651 : Unable to configure break on software breakpoints

Cause

The emulator controller is unable to execute the **bc -e bp** command, possibly because the emulator is in an unknown state or because of a hardware failure.

Action

Initialize the emulator or cycle power, then re-enter the command. If the same failure occurs, call your HP sales and service office.

Message 652 : Break condition must be specified

Cause

You entered **bc -e** or **bc -d** without specifying a break condition to enable or disable.

Action

Re-enter the **bc** command along with the enable/disable flag and the break condition you wish to modify.

Message 653 : Break condition configuration aborted.

Cause

Occurs when <CTRL> c is entered during **bc** display.

Action

None.

Message 661 : Software breakpoint break condition is disabled

Cause

You entered the **bp** command and options; however, the software breakpoint break condition is disabled.

Action

Enable the software breakpoint feature with **bc -e bp**, then enter the desired breakpoints with **bp**.

Message 663 : Specified breakpoint not in list: %s

Cause

You tried to enable a software breakpoint (**bp -e <ADDRESS>**) that was not previously defined. The string %s prints the address of the breakpoint you attempted to enable.

Action

Insert the breakpoint into the table and memory by typing **bp <ADDRESS>**.

Message 664 : Breakpoint list full; not added: %s

Cause

The software breakpoint table is already reached the maximum of 32 breakpoints. The breakpoint you just requested, with address %s, was not inserted.

Action

Remove breakpoints that are no longer in use with **bp -r <ADDRESS>**. Then insert the new breakpoint.

Message 665 : Enable breakpoint failed: %s.

Cause

System failure or target condition.

Action

Check memory mapping and configuration questions.

Message 666 : Disable breakpoint failed: %s.

Cause

System failure or target condition.

Action

Check memory mapping and configuration questions.

Message 667 : Breakpoint code already exists: %s

Cause

You attempted to insert a breakpoint with **bp <ADDRESS>**; however, there was already a software breakpoint instruction at that location which was not already in the breakpoint table.

Action

Your program code is apparently using the same breakpoint instruction as **bp**. If multiple breakpoint instructions are available on your processor, either change those in your program code or modify the one **bp** uses with your emulator's configuration options (**cf** command). If only one instruction is available, remove the breakpoints from your program code and use **bp** to insert breakpoints.

Message 668 : Breakpoint not added: %s

Cause

You tried to insert a breakpoint in a memory location which was not mapped or was mapped as guarded memory.

Action

Insert breakpoints only within memory ranges mapped to emulation or target RAM or ROM.

Message 669 : Breakpoint remove aborted.

Cause

Occurs when <CTRL> c is entered during a **bp -r** command.

Action

None.

Message 670 : Breakpoint enable aborted.

Cause

Occurs when <CTRL> c is entered during a **bp -e** command.

Action

None.

Message 671 : Breakpoint disable aborted.

Cause

Occurs when <CTRL> c is entered during a **bp -d** command.

Action

None.

Message 680 : Stepping failed.

Cause

Stepping has failed for some reason. For example in the HP 64742 68000 emulator, this message appears if the stack pointer is odd and you enter a step command.

Action

Usually, this error message will occur with other error messages. Refer to the descriptions of the accompanying error messages to find out more about why stepping failed.

Message 682 : Invalid step count: %s

Cause

You specified an non-cardinal value for a step count in the **s** command (such as entering **s 22.1**).

Action

Reenter the step command, using only cardinal values (positive integers) for the step count.

Message 684 : Failed to disable step mode.

Cause

System failure.

Action

Run performance verification (**pv** command).

Message 685 : Stepping aborted

Cause

This message is displayed if a break was received during a **s** (step) command with a stepcount of zero (0). The break could have been due to any of the break conditions in **bc** or a <CTRL> c break.

Action

None.

Message 686 : Stepping aborted; number steps completed: %d

Cause

This message is displayed if a break was received during a **s** (step) command with a stepcount greater than zero. The break could have been due to any of the break conditions in **bc** or a <CTRL> c break. The number of steps completed is displayed.

Action

None.

Message 688 : Step display failed.

Cause

System failure or target condition.

Action

Check memory mapping and configuration questions.

Message 689 : Break due to cause other than step.

Cause

An activity other than a **step** command caused the emulator to break. This could include any of the break conditions in a **bc** command or a <CTRL> c break.

Action

None.

Message 692 : Trace error during CMB execute.

Cause

System failure.

Action

Run performance verification (**pv** command).

Message 693 : CMB execute; run started

Cause

This status message is displayed when you are making coordinated measurements. The CMB /EXECUTE pulse has been received; the emulation processor started running at the address specified by the **rx** command.

Action

None; information only.

Message 694 : Run failed during CMB execute.

Cause

System failure or target condition.

Action

Run performance verification (**pv** command), and check target system.

Message 700 : Target memory access failed

Cause

This message is displayed if the emulator was unable to perform the requested operation on memory mapped to the target system.

Action

In most cases, the problem results from the emulator's inability to break to the monitor to perform the operation. See message 608.

Message 702 : Emulation memory access failed.

Cause

System failure.

Action

Run performance verification (**pv** command).

Message 707 : Request access to guarded memory: %s

Cause

The address or address range specified in the command included addresses within a range mapped as guarded memory. When the emulator attempts to access these during command processing, the above message is printed, along with the specific address or addresses accessed.

Action

Re-enter the command and specify only addresses or address ranges within emulation or target RAM or ROM. Or, you can remap memory so that the desired addresses are no longer mapped as guarded.

Message 710 : Memory range overflow.

Cause

On a word machine, typing "**m -dw 0ffff**" will cause a rounding error that overflows physical memory.

Action

Reduce memory display request.

Message 720 : Invalid map term number: %s

Cause

You attempted to delete a mapper term that does not exist. For example, you may have tried **map -d 8** on the 68000 emulator, which only has seven map terms. Or you may have tried **map -d 2**, when only one mapper term has been defined.

Action

Use the **map** command to determine the numbers of the terms currently mapped. Then delete the appropriate mapper term.

Message 721 : No map terms available; maximum number already defined

Cause

You tried to add more mapper terms than are available for this emulator. For example, with the 68000 emulator, there are only 7 terms. If you had already defined memory types for these terms, then tried to map another term, you would see the above error message.

Action

Either combine map ranges to conserve on the number of terms or delete mapper terms that aren't really needed to free another mapper term.

Message 723 : Invalid map address range: %s

Cause

You specified an invalid address range as an argument to the **map** command. For example, you may have specified digits that don't correspond to the base specified, or you forgot to precede a hexadecimal letter digit with a number, or the upper boundary of the range you specified is less than the lower boundary.

Action

Re-enter the **map** command and the address specification. See the <ADDRESS> syntax pages in the *Emulator User's Guide* and the <EXPRESSION> syntax pages in this manual for information on address specifications. Also, make sure that the upper boundary specification is greater than the lower boundary specification (the lower boundary must always precede the upper boundary on the command line).

Message 724 : Address not mappable: %s.

Cause

Trying to map an address that has a non-mappable function code. On 80C196, "**map 0..40@I eram**" is not mappable.

Action

Refer to the *Emulator User's Guide* for information on addresses which may have function codes.

Message 725 : Unable to load new memory map; old map reloaded.

Cause

There is not enough emulation memory left for this request.

Action

Reduce the amount of emulation memory requested.

Message 726 : Unable to reload old memory map; hardware state unknown.

Cause

System failure.

Action

Run performance verification (**pv** command).

Message 730 : Invalid memory map type: %s

Cause

You specified a memory type while mapping that is not one of the supported types: **eram, erom, tram, trom,** or **grd.**

Action

Re-enter the **map** command, specifying only one of the five types listed above.

Message 731 : Invalid memory map attribute: %s.

Cause

Newer HP 64700 emulators may use a feature called memory map attributes where an emulator specific item can be specified for any map term. If the item is not recognized it is an error. For example, the following command will cause the error: "map 0..100 eram bad_item".

Action

Refer to the *Emulator User's Guide* for information on valid memory map attributes.

Message 732 : Invalid memory type for 'other' range: %s

Cause

Most emulators restrict the memory types for **map other <type>** to **tram, trom,** or **grd.** If you see the above message, you have tried to map the "other" range to **eram** or **erom.**

Action

Map the "other" range to **tram, trom,** or **grd.** Refer to the *Emulator User's Guide* for particular restrictions regarding the "other" type for your emulator.

Message 734 : Map range overlaps with term: %d

Cause

You entered a map term whose address range overlaps with one already mapped. For example, you may have entered a term **map 1000..2fff eram,** then tried to enter a term **map 2000..3fff erom.**

Action

Re-enter the map term so that ranges do not overlap, or combine terms and change the memory type. If you are using an emulator

whose processor supports segmentation or function codes, it is possible that you did intend an overlap. If so, you need to more fully specify the correct segment or function code for each memory range. Refer to the <ADDRESS> syntax pages in the *Emulator User's Guide* for information.

Message 736 : Memory not mapped as emulation: %s.

Cause

This error occurs when a feature available only for emulation memory is attempted with target memory. For example, this error occurs when you attempt to perform coverage measurements (see the **cov** command) on target memory.

Action

You must remap the address range as emulation memory.

Message 738 : Unable to reset coverage bit data.

Cause

System failure.

Action

Run performance verification (**pv** command).

Message 740 : I/O port access failed

Cause

The emulator was unable to read or write the port specified in the **io** command. This message is also printed if your processor does not support separate I/O.

Action

If your processor does not support separate I/O, use the **m** command to modify I/O ports. Otherwise, retry the operation, and make sure that you are specifying a valid I/O address.

Message 750 : Copy image aborted; next destination: %s

752 : Copy memory aborted; next destination: %s

754 : Memory modify aborted; next address: %s

756 : Memory search aborted; next address: %s

758 : Coverage aborted; next address: %s

Cause

One of these message is displayed if a break occurs during processing of the **cim**, **cp**, **m**, **ser**, or **cov** commands, respectively. The break could result from any of the break conditions (except **bp**) or could have resulted from a <CTRL> c break.

Action

Retry the operation. If breaks are occurring continuously, you may wish to disable some of the break conditions with the **bc** command.

Message 800 : Invalid command: %s

Cause

You have entered a command which is not part of the standard Terminal Interface command set (documented in this manual) and was not found in the currently defined macros.

Action

Enter only commands defined in this manual or in the macro set. You can display the macro set using **mac**. You can rename commands or name command groups using the **mac** command.

Message 801 : Invalid command group: %s.

Cause

This error occurs when you specify an invalid group name in the **help -s <group>** command.

Action

Enter the **help** command with no options for a listing of the valid group names.

Message 802 : Invalid command format.

Cause

This error occurs when an invalid macro is entered, for example, **mac {help;{}**.

Action

Refer to the **mac** command description.

Message 807 : Macro list full; macro not added.

Cause

The maximum number of macros have been defined.

Action

You must delete macros before adding any new macros.

Message 809 : Macro buffer full; macro not added.

Cause

This error occurs when the memory reserved for macros is all used up.

Action

You must delete macros to reclaim memory in the macro buffer.

Message 812 : Invalid macro name: %s

Cause

You tried to delete a macro that did not exist; or you tried to define a new macro with a name containing characters other than letters, digits, or underscores.

Action

Use the **mac** command to display the names of macros in the macro table before deleting them with **mac -d <NAME>**. Define new macro names using only letters, digits, and underscore characters.

Message 813 : Command line too long; maximum line length: %d.

Cause

This error occurs when the command line exceeds the maximum number of characters.

Action

Split the command line into two command lines.

Message 814 : Command line too complex.

Cause

There was not enough memory for the expressions in the command line.

Action

Split up the command line, or use fewer expressions.

Message 815 : Missing macro parameter: %s

Cause

This error occurred because you did not include a parameter with the specified **mac** command for macro expansion.

Action

Enter the command again, and include the appropriate parameter for the macro expansion.

Message 816 : Command line too complex.

Cause

Too many expression operators are used.

Action

Split up the command line, or use fewer expressions.

Message 818 : Command line too complex.

Cause

A maximum nesting level has been exceeded for nested command execution.

Action

Reduce the number of nesting levels.

Message 820 : Unmatched quote encountered

Cause

In entering a string, such as with the **echo** command, you didn't properly match the string delimiters (either " or ""). For example, you might have entered

echo "set S1 to off

Action

Re-enter the command and string, making sure to properly match opening and closing delimiters. Note that both delimiters must be the same character. For example:**echo "set S1 to off"**.

Message 822 : Unmatched command group encountered

Cause

You entered the **mac** or **rep** command group without matching braces {}. For example: **mac test={rst -m;cf** or **rep 2 {rst -m;map**.

Action

Re-enter the command, making sure to match braces around commands you want grouped into the macro or repeat. For example: **mac test= {rst -m;cf}**.

Message 824 : Maximum number of arguments exceeded.

Cause

Exceeding the limit of 100 arguments per command.

Action

Reduce the number of arguments in the command.

Message 826 : Maximum argument buffer space exceeded.

Cause

Exceeding space limits for argument lists.

Action

Reduce request.

Message 840 : Invalid date: %s

Cause

You have specified the date format incorrectly in the **dt** command.

Action

Re-enter the command with the correct date format. Refer to the **dt** command syntax pages in this manual for the correct format.

Message 842 : Invalid time: %s

Cause

You have incorrectly specified the time format in the **dt** command.

Action

Re-enter the command with the correct time format. Refer to the **dt** command syntax pages in this manual for the correct format.

Message 844 : Invalid repeat count: %s

Cause

You entered a non cardinal value for the repeat count in the **rep** command, such as **rep 22.1 <command_group>**.

Action

Re-enter the **rep** command, specifying only a cardinal number (positive integer) for the repeat count.

Message 850 : Attempt to load code outside of allocated bounds.

Cause

This error occurs when the **lcd** command attempts to load an absolute file that contains code or data outside the range allocated for system code.

Action

Generally, you will not use the **lcd** command. The **lcd** command is intended to be used by high-level interfaces to the HP 64700.

Message 875 : Invalid syntax for global or user symbol name: %s

Cause

This error occurs when you enter a global or user symbol name with incorrect syntax.

Action

Make sure that you enter the global or user symbol name using the correct syntax. When specifying a global symbol, make sure that you precede the global symbol with a colon (for example, :glb_sym). When specifying a user symbol (created with the **sym** command), make sure that you enter the name correctly without a colon.

Message 876 : Invalid syntax for local symbol or module: %s

Cause

This error occurs when you enter a local symbol or module name with incorrect syntax.

Action

When entering a local symbol name using the **sym** command, make sure that you specify the module name, followed by a colon, then the symbol name (for example module:loc_sym). Make sure that you specify the module name correctly.

Message 877 : Symbol not found: %s

Cause

This occurs when you try to enter a symbol name that doesn't exist.

Action

Enter a valid symbol name.

Message 878 : Symbol cannot contain wildcard in this context.

Cause

You tried to enter a global, local, or user symbol name using the wildcard (*) incorrectly.

Action

When you enter the symbol name again, include the wildcard (*) at the end of the symbol.

Message 879 : Symbol cannot contain text after the wildcard.

Cause

You tried to include text after the wildcard specified in the symbol name (for example, sym*text).

Action

Enter the symbol again, but don't include text after the wildcard (*).

Message 880 : Conflict between expected and received symbol information.

Cause

The information you supplied in a symbol definition is not what the HP 64700 expected to receive.

Action

Make sure that all symbols in the symbol file are defined correctly. Verify that there are no spaces in the address definitions for the symbols in the symbol file being downloaded.

Message 881 : Ascii symbol download failed.

Cause

This error occurs because the system is out of memory.

Action

You must either reduce the number of symbols to be loaded, or free up additional system space and try the download again.

Message 882 : No module specified for local symbol.

Cause

This error occurs because you tried to specify a local symbol name without specifying the module name where the symbol is located.

Action

Enter the module name where the local symbol is located, followed by a colon, then the local symbol name.

Message 901 : Invalid firmware for emulation subsystem.

Cause

This error occurs when the HP 64700 system controller determines that the emulation firmware (ROM) is invalid.

Action

This message is not likely to occur unless you have upgraded the ROMs in your emulator. Be sure that the correct ROM is installed in the emulation controller.

Message 902 : Invalid analysis subsystem; product address: %s.

Cause

This error occurs when the HP 64700 system controller determines that the analysis firmware (ROM) is invalid.

Action

This message is not likely to occur unless you have upgraded the ROMs in your emulator. Be sure that the correct ROMs are installed in the analyzer board.

Message 903 : Invalid ET subsystem; product address: %s.

Cause

Detects an invalid ET. Used only internally.

Action

None.

Message 904 : Invalid auxiliary subsystem; product address: %s.

Cause

For future products.

Action

None.

Message 911 : Lab firmware for emulation subsystem.

Cause

This message should never occur. It shows that you have an unreleased version of emulation firmware.

Action

None.

Message 912 : Lab firmware analysis subsystem; product address: %s.

Cause

This message should never occur. It shows that you have an unreleased version of analysis firmware.

Action

None.

Message 913 : Lab firmware subsystem; product address: %s.

Cause

This message should never occur. It shows that you have an unreleased version of system controller firmware.

Action

None.

Message 914 : Lab firmware auxiliary subsystem; product address: %s.

Cause

This message should never occur. It shows that you have an unreleased firmware version of the auxiliary subsystem.

Action

None.

Analyzer Error Messages

Message 1000 : Conflicting disassembler option: <option>.

Cause

This error occurs when you attempt to specify inverse assembly options (tl -o<ialopts>) which are not allowed with each other.

Action

You must not use conflicting inverse assembly options in the same trace list command.

Message 1001 : Invalid disassembler option: <option>.

Cause

The <ialopts> option specified with the "**tl -o**" command is not valid.

Action

Refer to the appropriate *Emulator User's Guide* for a list of the valid options.

Message 1102 : Invalid bit range; crosses two multiples of 16: <sig#>..<sig#>.

Cause

This error occurs when defining trace labels. A trace label may not contain trace signals crossing two 16-bit boundaries. For example, the command "**tlb name 1..32**" will cause this error because "name" contains signals which cross the 15-16 and 31-32 16-bit boundaries.

Action

Redefine your trace label so that no more than one 16-bit boundary is crossed.

Message 1103 : Invalid bit range; out of bounds: <sig#>..<sig#>.

Cause

This error occurs when defining trace labels, and you have attempted to assign non-existent trace signals to a label.

Action

Enter the trace activity command to view the trace signals present, and use only these signals when defining trace labels.

Message 1104 : Invalid bit range; too wide: <sig#>..<sig#>

Cause

This error occurs when defining trace labels, and you have attempted to assign more than 32 trace signals to a label.

Action

Use more than one trace label to define over 32 trace signals.

Message 1105 : Unable to delete label; used by emulation analyzer: <label>.

Cause

This error occurs when you attempt to delete an emulation trace label which is currently being used as a qualifier in the emulation trace specification or is currently specified in the emulation trace format.

Action

Display the emulation trace sequencer specification in the easy configuration, display the emulation trace patterns in the complex configuration, or display the trace format to see where the label is used. Also, you should check **tcq** and **tpq** for uses of that label. You must change the pattern or format specification to remove the label before you can delete it.

Message 1106 : Unable to delete label; used by external state analyzer: <label>.

Cause

This error occurs when you attempt to delete an external trace label which is currently being used as a qualifier in the external state trace specification or is currently specified in the external trace format.

Action

Display the external trace sequencer specification in the easy configuration, display the external trace patterns in the complex configuration, or display the external trace format to see where the label is used. Also, check **tcq** and **tpq** for uses of that label. You must change the pattern or format specification to remove the label before you can delete it.

Message 1107 : Unable to delete label; used by external timing analyzer: <label>.

Cause

This error occurs when you attempt to delete an external trace label which is currently being used as a qualifier in the external timing trace specification.

Action

Remove the label from the external timing analyzer specifications, and then delete the label.

Message 1108 : Unable to redefine label; used by emulation analyzer: <label>.

Cause

This error occurs when you attempt to redefine an emulation trace label which is currently used as a qualifier in the emulation trace specification.

Action

Display the emulation trace sequencer specification in the easy configuration, display the emulation trace patterns in the complex configuration, or display the emulation trace format to see where the label is used. You must change the pattern or format specification to remove the label before you can redefine it.

Message 1109 : Unable to redefine label; used by external state analyzer: <label>.

Cause

This error occurs when you attempt to redefine an external trace label which is currently used as a qualifier in the external state trace specification.

Action

Display the external trace sequencer specification in the easy configuration, or display the external trace patterns in the complex configuration to see where the label is used. You must change the pattern or format specification to remove the label before you can redefine it.

Message 1110 : Unable to redefine label; used by external timing analyzer: <label>.

Cause

This error occurs when you attempt to redefine an emulation or external trace label which is currently being used as a qualifier in the external timing trace specification.

Action

Remove the label from the external timing analyzer specifications, and then redefine the label.

Message 1111 : Unable to redefine label; belongs to external analyzer: <label>.

Cause

This error occurs when you attempt to redefine an external analyzer label with the emulation trace label command (for example, tlb xbits 0..16).

Action

Either use a different label name, or delete the external analyzer label before defining a label of the same name for the emulation analyzer.

Message 1112 : Unable to redefine label; belongs to emulation analyzer: <label>.

Cause

This error occurs when you attempt to redefine an emulation analyzer label with the external trace label command (for example, xtlb addr 0..19).

Action

Either use a different label name, or delete the emulation analyzer label before defining a label of the same name for the external analyzer.

Message 1114 : Label belongs to external analyzer: <label>.

Cause

When the external analyzer is in an independent mode, this error occurs when you attempt to use an external analyzer label in an emulation trace command (for example, tg xlabel=0).

Action

Only use external trace labels in external trace commands (when the external analyzer is in an independent mode).

Message 1115 : Label belongs to emulation analyzer: <label>.

Cause

When the external analyzer is in an independent mode, this error occurs when you attempt to use an emulation analyzer label in an external trace command (for example, xtg addr=5).

Action

Only use emulation trace labels in emulation trace commands (when the external analyzer is in an independent mode).

Message 1130 : Illegal base for count display.

Cause

When specifying the trace format, counts may only be displayed relative or absolute. When counting states, the count is always displayed as a decimal number.

Action

Respecify the trace format without using a base for the count column. Also, you can use ",**A**" to specify that counts be displayed absolute, or you can use ",**R**" to specify that counts be displayed relative.

Message 1131 : Illegal base for mnemonic disassembly display.

Cause

When specifying the trace format, you cannot specify a number base for the column containing mnemonic information.

Action

Respecify the trace format without using a base for the mnemonic column.

Message 1132 : Illegal base for sequencer display.

Cause

When specifying the trace format, you cannot specify a number base for the column containing sequencer information.

Action

Respecify the trace format without using a base for the sequencer column.

Message 1133 : Trace format command failed; using old format.

Cause

This error occurs when the trace format command fails for some reason. This error message always occurs with another error message.

Action

Refer to the "Action" description for the other error message displayed.

Message 1137 : Mnemonic disassembly not supported for external trace.

Cause

This error occurs when you attempt to specify a mnemonic information column in the external trace format. There is no mnemonic disassembly for the external trace.

Action

Respecify the trace format without the mnemonic column.

Message 1138 : Illegal width for symbol display: %s

Cause

This error occurs when the value specified for the trace format address field width is not valid.

Action

Enter the **tf** command again, and specify the width of the address field for symbol display within the range of 4 to 55.

Message 1139 : Illegal width for addr display, mne not specified.

Cause

This error occurs when you specify a width for the address field in the **tf** command, but do not include the **mne** option.

Action

Enter the command again, and include the **mne** option.

Message 1140 : Symbol display unsupported.

Cause

This error occurs when you try to display symbols in the trace list, but the emulator you are using doesn't support symbols.

Action

Enter the **tl** command again, but don't try to display symbols.

Message 1141 : Symbol display unavailable without mne field.

Cause

This error occurs when you try to display symbols, but have not included the **mne** option to the **tf** command.

Action

Don't try to display symbols unless the **mne** field has already been specified.

Message 1202 : Trigger position out of bounds: <bounds>.

Cause

This error occurs when you attempt to specify a number of lines to appear either before or after the trigger which is greater than the number of lines allowed. The <bounds> string indicates the incorrect range that you typed (not the correct limits on the range).

Action

Be sure that the trigger position specified is within the range -1024 to 1023.

Message 1207 : Invalid clock channel: <name>.

Cause

Valid clock channels are L, M, and N. If you have an external analyzer, the J and K channels are also valid.

Action

Respecify the command using valid clock channels.

Message 1209 : Operator must be "and" or "or": <expression>.

Cause

When combining trace labels to specify trace patterns (in simple expressions or with the **tpat** command), an operator of either "and" or "or" must appear between the label qualifiers.

Action

Refer to the "Expressions in Trace Commands" section of the "Getting Started" chapter of the *Analyzer User's Guide* for information on valid patterns. Also refer to the "Expressions" chapter of the *Reference* manual.

Message 1210 : Illegal mix of = and !=.

Cause

When combining trace labels to specify patterns (in simple expressions or with the tpat command), all labels must either be equal to values or not equal to values.

Action

Refer to the "Expressions in Trace Commands" section of the "Getting Started" chapter of the *Analyzer User's Guide* for information on valid patterns. Also refer to the "Expressions" chapter of the *Reference* manual.

Message 1211 : Illegal mix of and/or.

Cause

When combining trace labels to specify patterns (in simple expressions or with the tpat command), all label qualifiers must either be ANDed together or ORed together. You cannot mix these operators.

Action

Refer to the "Expressions in Trace Commands" section of the "Getting Started" chapter of the *Analyzer User's Guide* for information on valid patterns. Also refer to the "Expressions" chapter of the *Reference* manual.

Message 1212 : Conflict with overlapping label: <label>.

Cause

When combining trace labels to specify patterns (in simple expressions or with the tpat command), you cannot combine labels which are defined for common trace signals. For example, the following easy configuration commands will result in this error: tlb low8 0..7; tlb low16 0..15; tg low8=0 and low16=1.

Action

Either omit one of the overlapping labels, or redefine your labels so that they do not contain common trace signals. You could also circumvent this error by using don't cares in the appropriate places; for the example shown in cause, you could specify patterns **tg low8=0xx0xY and low16=1**.

Message 1213 : Illegal mix of !=/and.

Cause

When combining trace labels to specify patterns (in simple expressions or with the tpat command), labels which are not equal to values must be ORed together so that the entire pattern specifies a "not equals" condition.

Action

Refer to the "Expressions in Trace Commands" section of the "Getting Started" chapter of the *Analyzer User's Guide* for information on valid patterns. Also refer to the "Expressions" chapter of the *Reference* manual.

Message 1214 : Illegal mix of =/or.

Cause

When combining trace labels to specify patterns (in simple expressions or with the tpat command), labels which are equal to values must be ANDed together so that the entire pattern specifies an "equals" condition.

Action

Refer to the "Expressions in Trace Commands" section of the "Getting Started" chapter of the *Analyzer User's Guide* for information on valid patterns. Also refer to the "Expressions" chapter of the *Reference* manual.

Message 1215 : Comparator must be = or !=: <label>.

Cause

When combining trace labels to specify patterns (in simple expressions or with the tpat command), the value of the label can only be specified with the "=" or "!=" operators.

Action

Refer to the "Expressions in Trace Commands" section of the "Getting Started" chapter of the *Analyzer User's Guide* for information on valid patterns. Also refer to the "Expressions" chapter of the *Reference* manual.

Message 1217 : Illegal pattern name: <name>.

Cause

Valid pattern names are p1 through p8.

Action

Use only valid pattern names.

Message 1218 : Illegal comparator for range qualifier: !=.

Cause

When specifying a range with the trng command, you cannot use the"!=" operator.

Action

Use the "!r" range name.

Message 1219 : Range cannot be combined with any other qualifier.

Cause

For example, the following easy configuration command will result in this error: tsto addr=400..4ff and data=40.

Action

Do not attempt to combine labels when using range qualifiers.

Message 1221 : Range resource in use.

Cause

This error occurs when you attempt to use two different range expressions in the "easy" configuration trace specification or when you attempt to redefine the "complex" configuration range resource while it is currently being used as a qualifier in the trace specification.

Action

Only one range expression may be used in the "easy" configuration trace specification. In the "complex" configuration, display the sequencer specification to see where the range resource is being used and remove it; then, you can redefine the range resource.

Message 1224 : Sequence term number out of range: <term>.

Cause

This error occurs when a sequencer qualification command (**tif**, **telif**, **tsq**, or **tsto**) specifies a non-existent sequence term. The easy configuration sequencer may have a maximum of 4 sequence terms. Eight sequence terms exist in the complex configuration sequencer.

Action

Re-enter the command using an existing sequence term.

Message 1225 : Sequence term not contiguous: <term>.

Cause

This error occurs when you attempt to insert a sequence term which is not between existing terms or after the last term. For example, the following easy configuration commands will result in this error: tg any; tsq -i 4.

Action

Be sure that the sequence term you enter is either between existing sequence terms or after the last sequence term.

Message 1226 : Too many sequence terms.

Cause

This error occurs when you attempt to insert more than 4 sequence terms.

Action

Do not attempt to insert more than 4 sequence terms.

Message 1227 : Sequence term not defined: <term>.

Cause

This error occurs when you attempt to delete, or specify a primary branch expression for, a sequence term number which is possible, but which is not currently defined.

Action

Insert the sequence term, and respecify the primary branch expression for that term.

Message 1228 : One sequence term required.

Cause

This error occurs when you attempt to delete terms from the sequencer when only one term exists.

Action

At least one term must exist in the sequencer. Do not attempt to delete sequence terms when only one exists.

Message 1234 : Invalid occurrence count: <number>.

Cause

Occurrence counts may be from 1 to 65535.

Action

Re-enter the command with a valid occurrence count.

Message 1235 : Illegal threshold value: <value>.

Cause

Threshold voltage specifications may be from -6.4 V to +6.35 V in increments of 0.05 V.

Action

Re-enter the command with a valid threshold voltage.

Message 1237 : Option specified more than once: <option>.

Cause

When specifying external threshold voltages, this error occurs when you attempt to specify the threshold voltage for either the upper or lower byte twice.

Action

You must re-enter the command so that the threshold voltage is only specified once for each option (upper or lower byte).

Message 1239 : Clock speed not available with current count qualifier.

Cause

This error occurs when you attempt to specify a fast (F) or very fast (VF) maximum qualified clock speed when counting time (tcq time). This error also occurs when you attempt to specify a very fast (VF) maximum qualified clock speed when counting states (for example, tcq addr=400).

Action

Change the count qualifier; then, re-enter the command.

Message 1240 : Count qualifier not available with current clock speed.

Cause

This error occurs when you attempt to specify the "time" count qualifier when the current maximum qualified clock speed is fast (F) or very fast (VF). This error also occurs when you attempt to specify a "state" count qualifier when the maximum qualified clock speed is fast (F).

Action

Change the clock speed; then, change the count qualifier.

Message 1241 : Invalid qualifier resource or operator: <expression>.

Cause

When specifying complex expressions, you have either specified an illegal pattern or used an illegal operator.

Action

Refer to the "Complex Expressions" section of the "Accessing Full Analyzer Capability" chapter in the *Analyzer User's Guide* for information on valid patterns and operators. Also refer to the "Expressions" chapter in the *Reference* manual.

Message 1245 : Range qualifier not accessible in easy configuration.

Cause

This error occurs when you attempt to use the trng command in the easy configuration.

Action

Changing into the complex configuration will allow you to use the trng command; otherwise, specify the range in easy configuration command expressions.

Message 1246 : Pattern qualifiers not accessible in easy configuration.

Cause

This error occurs when you attempt to use the tpat command in the easy configuration.

Action

Changing into the complex configuration will allow you to use the tpat command; otherwise, specify the patterns in easy configuration command expressions.

Message 1248 : Range term used more than once

Cause

This error occurs when you attempt to use the range resource more than once in a sequencer branch expression.

Action

You cannot use the range resource more than once in a sequencer branch expression.

Message 1249 : Invalid qualifier expression: <expression>.

Cause

This error message is shown with the errors that occur when patterns, the range, or the arm condition is used more than once within a set. This error message also occurs when intraset operators are not the same. For example, the following complex expression will result in this error: p1 ~ p2 | p3.

Action

Refer to the "Complex Expressions" section of the "Accessing Full Analyzer Capability" chapter in the *Analyzer User's Guide* for information on valid patterns and operators. Also refer to the "Expressions" chapter in the *Reference* manual.

Message 1250 : Arm term used more than once

Cause

This error occurs when you attempt to use the "arm" qualifier more than once in a sequencer branch expression.

Action

You cannot use the "arm" qualifier more than once in a sequencer branch expression.

Message 1251 : Trigger term cannot be term 1.

Cause

This error occurs when to attempt to specify the first sequence term as the trigger term. The trigger term may be any term but the first.

Action

Respecify the trigger term as any other sequence term.

Message 1253 : Invalid pod number: <pod#>.

Cause

This error message occurs when you attempt to specify a slave clock for a non-existent analyzer pod.

Action

Use the trace activity command to display the valid pod numbers, and use only these numbers when entering commands.

Message 1257 : Pod belongs to external analyzer: <pod#>.

Cause

This error occurs when you attempt to specify a slave clock for the external analyzer pod with the emulation analyzer's trace slave clock command. This error only occurs when the external analyzer is in its independent state mode.

Action

Use the external trace slave clock command to specify a slave clock for the external analyzer pod.

Message 1300 : Incompatible external trace mode.

Cause

This error message occurs when you attempt to use an external trace command (other than **xtv, xtlb**, or **xtmo**) while the external analyzer is aligned with the emulation analyzer. The message is also display if you attempt to use external state trace commands when the external analyzer is in timing mode; or if you attempt to use external timing trace commands when the external analyzer is in state mode.

Action

Change the external trace mode, and re-enter the command.

Message 1301 : External label in use: <label>.

Cause

This error occurs when you attempt to select the external analyzer's independent state mode while an external trace label is currently used as a qualifier in the emulation analyzer trace specification.

Action

Remove any external trace label qualifiers from emulation trace specifications before selecting the external analyzer's independent state mode.

Message 1302 : Trig1 signal cannot be driven and received.

Cause

This error occurs when you attempt to specify the internal trig1 signal as the trace arm condition while the same analyzer's trigger output is currently driving the trig1 signal. This error also occurs if you attempt to specify that the trigger output drive the internal trig1 signal while that signal is currently specified as the arm condition for the same analyzer.

Action

You can either change the arm or the trigger output specification; in either case, make sure that they do not use the same internal signal.

Message 1303 : Trig2 signal cannot be driven and received.

Cause

This error occurs when you attempt to specify the internal trig2 signal as the trace arm condition while the same analyzer's trigger output is currently driving the trig2 signal. This error also occurs if you attempt to specify that the trigger output drive the internal trig2 signal while that signal is currently specified as the arm condition for the same analyzer.

Action

You can either change the arm or the trigger output specification; in either case, make sure that they do not use the same internal signal.

Message 1304 : Analyzer trace running.

Cause

This error occurs when you attempt to change the external analyzer mode while a trace is in progress.

Action

Halt the trace before changing the external analyzer mode.

Message 1305 : CMB execute; emulation trace started.

Cause

This status message informs you that an emulation trace measurement has started as a result of a CMB execute signal (as specified by the "**tx -e**" command).

Message 1306 : CMB execute; external trace started.

Cause

This status message informs you that an emulation trace measurement has started as a result of a CMB execute signal (as specified by the "**xtx -e**" command).

Message 2021 : Period not in 1/2/5 sequence: <period>.

Cause

This error message occurs when the external timing sample period is not in a 1/2/5 sequence; for example, 10ns, 20ns, 50ns, 100ns, 200ns, 500ns, 1us, 2us, 5us, etc. Some examples of invalid sample period specifications are: 12ns, 18ns, 25ns, 60ns, 80ns, etc.

Action

Use a number in the 1/2/5 sequence when specifying the external timing sample period.

Message 2022 : Sample period out of bounds: <bounds>.

Cause

The external timing sample period must be between 10 ns and 50 ms (in a 1/2/5 sequence).

Action

Re-enter the command with the sample period between the bounds shown.

Message 2030 : Negated patterns not allowed in timing.

Cause

This error occurs when you attempt to specify a "not equals" expression when defining the external timing trigger. You can only specify labels which equal patterns (of 1's, 0's, or X's).

Action

Do not attempt to specify negated timing patterns.

Message 2031 : Invalid trigger duration: <duration>.

Cause

This error occurs when you attempt to specify an external timing trigger duration which is in the valid range but is not a multiple of 10 ns.

Action

Re-enter the command with the trigger duration as a multiple of 10 ns.

Message 2032 : Trigger duration out of bounds: <bounds>.

Cause

This error occurs when you attempt to specify an external timing trigger duration outside the valid range. A "greater than" duration must fall within the range of 30 ns to 10 ms (and must be a multiple of 10 ns). A "less than" duration must fall within the range 40 ns to 10ms (and must be a multiple of 10 ns).

Action

Re-enter the command with the trigger duration within the bounds shown.

Message 2042 : Trigger delay out of bounds: <bounds>.

Cause

This error occurs when you attempt to specify an external timing trigger delay outside the valid range. The external timing trigger delay must be between 0 and 10 ms (in 10 ns increments).

Action

Re-enter the command with the trigger delay within the bounds shown.

Command Entry

The HP 64700-Series Emulators Terminal Interface provides several features to ease command entry and simplify recognition of the current emulator state.

Note

The backspace key (or <CTRL> h, or DEL) will only backspace to the start of the current line; this is true even if the command wraps to more than one line.

Prompts

The HP 64700-Series Emulators use a variety of prompt characters to describe the current emulation status. These are:

R	Reset state from emulation system, resulted from the **rst** command or another emulator command which resets the emulator. **Not** a target system reset.
U	Running user program, resulted either from the **r** command or /EXECUTE line asserted on the CMB during a synchronized measurement.
M	Running monitor program.
W	Waiting for CMB to become READY. (See the *CMB User's Guide.*)

T	Waiting for target system reset to complete a **r rst** command. (Will be shown only if **r rst** is supported, see your *Emulator User's Guide*.)
?	Unknown state.

The following prompt characters may or may not be used by your emulator:

c	Slow or no clock input from target system (displayed only if **clk=ext** in configuration).
r	Emulator is reset by target system.
h	Processor halted (by program or target system).
i	Processor in idle state.
g	Bus grant to target system device.
b	Slow or no emulation processor bus cycles.

If multiple conditions occur which would call for more than one prompt character, the priority of characters is as follows:

c R r h i g b U M W T ?

Note that the prompt character is unrelated to the prompt string, which is set to **>** at powerup and may be modified using the **po** command. The current prompt string is concatenated after the prompt character.

Command Line Editing

You can enable command line editing to include the ability to manipulate command text lines. When command line editing is enabled, a subset of the Korn shell (ksh) **vi** editing mode features is implemented.

Command line editing has two typing modes. The normal command entry is input mode. The input mode functions like normal (canonical) command entry. The control mode allows command modification.

Input Mode

The input mode allows for the use of the following:

<DEL>	This deletes the previous character.
<CTRL> r	This recalls the commands in the order last to first.
<CTRL> b	This recalls the commands in the order first to last.

Note

The recall **<CTRL> r** and **<CTRL> b** functions are the same as in standard (non-command line editing) mode except that the cursor is positioned at the start of the line instead of the end of the line.

Control Mode

Enter the control mode by entering the ESC (033 octal) character.

Command Search

You can locate previously entered commands using the following:

k	This option allows you to retrieve the previous command. Each successive **k** command accesses the next earlier command in the history list.

j	This retrieves the next command. Each successive **j** command accesses the next later command in the history list.
/<string>	This allows you to find a previous command in the history list matching <string>.
n	This repeats a previous match search for matches earlier in the history list.
N	This repeats a previous match search for matches later in the history list.

Cursor Movement

You can move the cursor using control mode commands

l	This moves the cursor forward (right) one character.
h	This moves the cursor back (left) one character.
$	This moves the cursor to the end of the line.
0 or ^	This moves the cursor to the start of the line.

Command Modification

You edit the command using the control mode commands

i	This inserts the cursor before the current position.
a	This inserts the cursor after the current position.
A	This allows you to append text to the end of the line.
r	This allows you to replace the current character.

x	This deletes the current character.
dd	This deletes the current line.
D	This deletes text from the current character to the end of the line.

Changing Modes

Use the **cl** command to display and change the command line editing mode. The command

```
cl -e
```

enables command line editing, while

```
cl -d
```

disables command line editing. The default condition is commmand line editing disabled.

Command Abort

You can abort any command's execution by typing:

```
<CTRL> c
```

You will generally use this to stop **rep 0 <COMMAND>** repeats.

Command Recall

The Terminal Interface provides you with two methods to recall up to 16 previous commands, which will save you typing time.

`<CTRL> r` recalls the previous command (last entered is recalled first, as in a stack).

`<CTRL> b` recalls commands in backwards order (first entered is recalled first, as in a queue).

For example, assume you entered the following equates:

```
R> equ b=1
R> equ c=2
```

Pressing **<CTRL> r** would display

```
R> equ c=2
```

Pressing **<CTRL> b** would display

```
R> equ b=1
```

Multiple Commands

You may issue multiple commands on the same command line by separating them with the ; (semicolon) character. For example:

```
R> rst -m; map; cf
```

If you are using multiple commands in the **mac** (macro) or **rep** (repeat) commands, they need to be enclosed in braces ({}) or they will be misinterpreted. For example, using the sequence above:

```
M> rep 2 {rst -m;map;cf}
```

This sequence will repeat the command sequence in the braces twice. Now look at the following:

```
M> rep 2 rst -m;map;cf
```

Here, only the **rst -m** command will be repeated; then the **map** command will execute once and the **cf** command will execute once.

Commenting

You may include comments at the end of any command line using the **#** character. Although these comments are not saved with the command status, they can be useful to you in building a command file to be saved on a host and downloaded to the emulator at a later time. For example, you might want to comment some equates in a command file. Your file could look like this:

```
equ start=2000 #beginning of program
equ clear=200c #clears input port
equ readinput=2010 #routine reading input port
equ messagelength=17T #length in bytes of each output message
```

Notes

Index

Notes

Hewlett-Packard
Printed in the USA