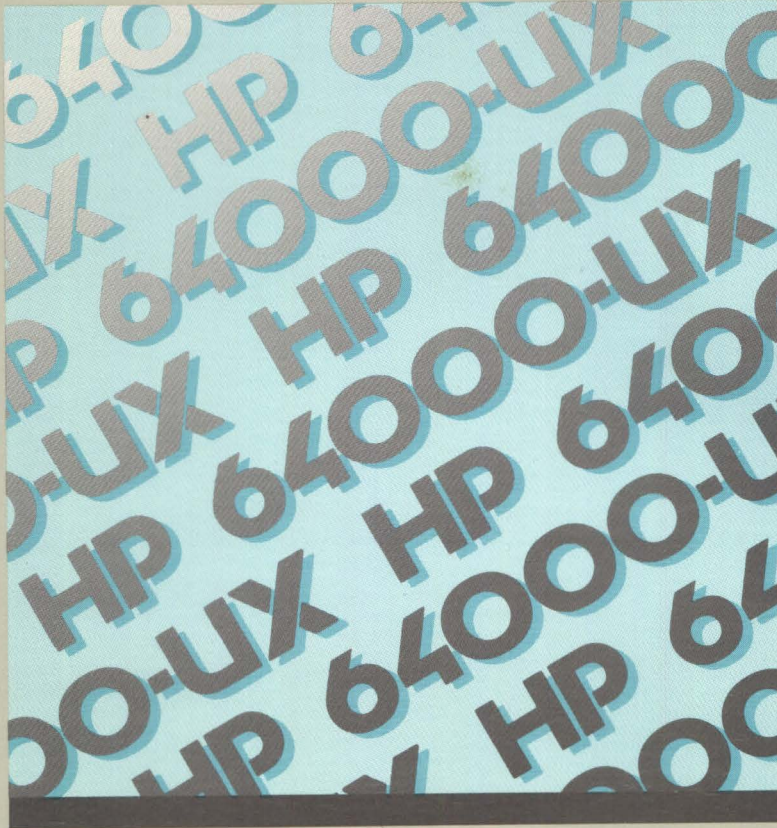


# HEWLETT-PACKARD



CASE SOLUTIONS FOR MICROPROCESSORS

HP 64430  
**68030 Emulator  
User's Guide**

*DesignCenter*

---

HP 64430

**68030**

**Emulator**

**User's Guide**



HP Part No. 64430-97000

Printed in U.S.A.

February 1990

Edition 1



---

## Certification and Warranty

### **Certification**

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

### **Warranty**

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

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

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

### **Limitation of Warranty**

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

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

**Exclusive Remedies**

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

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

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

---

## Notice

**Hewlett-Packard makes no warranty of any kind with regard to this material, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose.**

Hewlett-Packard shall not be liable for errors contained herein or for incidental or consequential damages in connection with the furnishing, performance, or use of this material.

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

Copyright 1990 Hewlett-Packard Company.

This document contains proprietary information, which is protected by copyright. All rights are reserved. No part of this document may be photocopied, reproduced or translated to another language without the prior written consent of Hewlett-Packard Company. The information contained in this document is subject to change without notice.

HP and HP-UX are trademarks of Hewlett-Packard Company.

UNIX is a registered trademark of AT&T.

TORX is a registered trademark of Camcar Division of Textron, Inc.

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

---

## Printing History

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

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

**Edition 1**

**64430-97000, February, 1990**

# Electromagnetic Interference

---

---

## **What Is Electromagnetic Interference?**

All types of electronic equipment are potential sources of unintentional electromagnetic radiation which may cause interference with licensed communication services. Products which utilize digital waveforms such as any computing device are particularly characteristic of this phenomena and use of these products may require that special care be taken to ensure that Electromagnetic Interference (EMI) is controlled. Various government agencies regulate the levels of unintentional spurious radiation which may be generated by electronic equipment. The operator of this product should be familiar with the specific regulatory requirement in effect in his locality.

The HP 64000-UX has been designed and tested to the requirements of the Federal Republic of Germany VDE 0871 Level A. They have been licensed with the German ZZF as Level A products (FTZ C-112/82. These specifications and the laws of many other countries require that if emissions from these products cause harmful interference with licensed radio communications, that the operator of the interference source may be required to cease operation of the product and correct the situation.



---

## Reducing the Risk Of EMI

1. Ensure that the top cover of the HP 64120A Instrumentation Cardcage is properly installed and that all screws are tight (do not over tighten).
2. When using a feature set which includes cables that egress from the chassis slot of the HP 64120A, insure that the knurled nuts and ferrels, or brackets that ground the cable shields are clean and tight (Do not overtighten). The EEPROM Programmer cable has an exposed shield that must make contact with the cable clamp.
3. During times of infrequent use, disconnect the EEPROM Programmer and cables from the card cage and the target system.
4. Use only shielded coaxial cables on the four external BNC connectors on the rear of the HP 64120A
5. Use only the shielded IMB cable supplied with the HP 64120A for connection to additional HP 64120A Instrumentation Cardcages.
6. Use only shielded cables on the IEEE 488 interface connector to the host computer.

---

## Reducing Interference

In the unlikely event that emissions from the HP 64000-UX System result in electromagnetic interference with other equipment, you may use the following measures to reduce or eliminate the interference.

1. If possible, increase the distance between the emulation system and the susceptible equipment.
2. Rearrange the orientation of the chassis and cables of the emulation system.
3. Plug the HP 64120A into a separate power outlet from the one used by the susceptible equipment (the two outlets should be on different electrical circuits).
4. Plug the HP 64120A into a separate isolation transformer or power line filter.

You may need to contact your local Hewlett-Packard sales office for additional suggestions. Also, the U.S.A. Federal Communications Commission has prepared a booklet entitled *How to Identify and Resolve Radio - TV Interference Problems* which may be helpful to you. This booklet (stock #004-000-00345-4) may be purchased from the Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402 U.S.A.

---

## Manufacturer's Declarations

### **U.S.A. Federal Communications Commission**

Warning - This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

### **Federal Republic of Germany**

Wenn Ihr Gerät in der Bundesrepublik Deutschland einschl. Westerin betrieben wird, senden Sie bitte die beiliegende Postkarte ausgefüllt an Ihr zuständiges Fernmeldeamt.

---

## Safety

### Summary of Safe Procedures

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

#### Ground The Instrument

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

#### Do Not Operate In An Explosive Atmosphere

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

#### Keep Away From Live Circuits

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

### **Do Not Service Or Adjust Alone**

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

### **Do Not Substitute Parts Or Modify Instrument**

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

### **Dangerous Procedure Warnings**

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

**Warning**



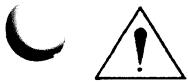
---

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

---

## Safety Symbols Used In Manuals

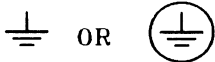
The following is a list of general definitions of safety symbols used on equipment or in manuals:



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



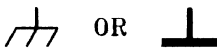
Indicates dangerous voltage (terminals fed from the interior by voltage exceeding 1000 volts must be marked with this symbol).



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



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



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



Alternating current (power line).



Direct current (power line).



Alternating or direct current (power line).

## Note



---

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

---

## Caution



---

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

---

## Warning



---

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

---

# Notice

---

**Caution**



---

**CONDUCTIVE FOAM OR PLASTIC OVER EMULATOR PINS MAY CAUSE ERRATIC OPERATION.**

---

The emulator user assembly pins are covered at the time of shipment with either a conductive foam wafer or a conductive plastic pin protector. This is done for two reasons:

- 1) to protect the user interface circuitry within the emulator from electro-static discharge (ESD),
- 2) to protect the delicate gold plated pins of the probe assembly from damage due to impact.

Both the foam and plastic protection devices are conductive. This may cause erratic performance of the emulation or analysis system during operation, and also during option\_test performance verification. Therefore, it is recommended that the foam or plastic device be removed before using the emulation or analysis system or before running option\_test performance verification.

When not using the emulator, the foam or plastic assembly should be replaced to retain protection for the probe pins and protection from ESD.

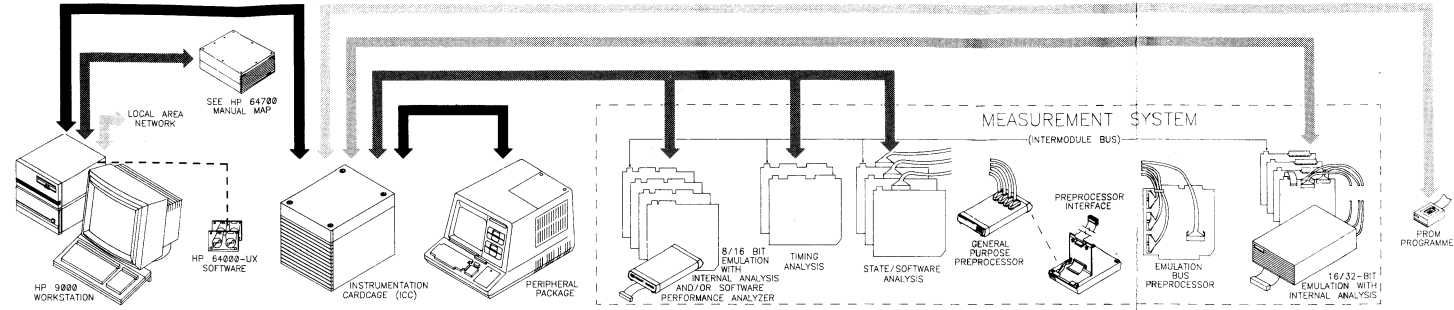
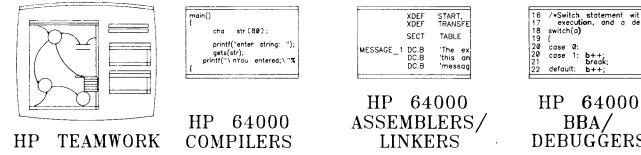


---

## Notes

SOFTWARE DEVELOPMENT TOOLS

HW & SW INTEGRATION TOOLS



PRE-INSTALLATION

HW & SW INSTALLATION

USE

TEST AND REPAIR

If All Else Fails:

HP 64000-UX MANUAL MAP

Effective for HP 64000-UX Microprocessor Development System  
Manuals printed after December 1986.

HOW TO USE THIS MAP

Tasks are listed down the left-hand side of the rows. Hardware and software are shown along the top of the columns. Row/column intersections show the manuals you need when performing the tasks using the hardware and software.  
EXAMPLE: All manuals needed to use an emulator are shown at the intersection of the USE row and EMULATION column. Further, the emulation reference manual is in the same binder with the processor-specific manual for your emulator.

KEY

- = Use the manual(s) pointed to by the arrow when performing the task (on the left) with the product (shown above).
- = Indicates manuals contained in same binder.
- = HP 9000 Reference Manuals may need to be used.
- = Software Installation Manual includes information for the product shown above.
- = Your friendly, fast, efficient, reliable, results-oriented HP Customer Support staff.
- = For some emulators, the contents of the two manuals shown here are combined into one manual.
- = Automated version for hardware and software system installation.
- Conversion Kit = Makes an existing HP 64100A or HP 64110A station into an HP 64000-UX supported package consisting of a terminal, cardcage, and flexible disc drives (if drives were part of the original product).
- Peripheral Package Operation = Shows you how to operate the parts of your converted HP 64100A or HP 64110A station.

REV. 3 4/89

# USING THIS MANUAL

---

## Organization

- Chapter 1** **Introducing The 68030 Emulator** contains a brief description of the 68030 emulator.
- Chapter 2** **Installing Emulation Hardware** contains information on installing your 68030 emulation system hardware into the instrumentation cardcage and making a measurement system. This chapter also contains information on connecting the emulator to your target system.
- Chapter 3** **Getting Started** steps you through the emulation process from creating an example program to performing measurements on the execution of that program in emulation.
- The **Getting Started** chapter discusses preparing your program modules and the files that are generated by assembling, compiling, and linking programs. See the appropriate cross assembler/linker and compiler manuals for more detailed information on preparing program modules for emulation.
- Chapter 4** **Configuring Your Emulator** shows how to access the emulation configuration questions, describes the options available when configuring the emulator, and shows how to load configuration command files from a previous emulation session.
- Chapter 5** **DeMMUer - What It Is And How It Works** describes what the deMMUer is, how the deMMUer operates, when to use the deMMUer, and the restrictions you need to observe when using the deMMUer.

- Chapter 6** **Target System Interface** provides information about the 68030 pins and how the emulator interacts with those pins. It also provides guidelines for using the emulator with a target system and provides information you need to know about how the emulator interacts with your target system.
- Chapter 7** **The Emulation Monitor Program** provides a detailed description of the emulation monitor program and how to modify it for your system requirements.
- Chapter 8** **Using Custom Coprocessors** describes how to make a custom coprocessor register format file and how to modify the emulation monitor so that your emulation system can display and modify coprocessor registers.
- Chapter 9** **Using Simulated I/O And Simulated Interrupts** describes how to set up your emulator to use host I/O resources to simulated target system I/O and how to use the simulated interrupt features of the emulator.
- Chapter 10** **How The Emulator Works** provides a detailed description of how many of the emulator features work. Understanding how the emulator works helps you use the emulator more effectively and helps you resolve problems you may encounter.
- Appendix A** **Emulation Error Messages** contains descriptions of the most serious error messages you may encounter and information on how to correct the errors.
- Appendix B** **Demonstration Source Files** provides listings of the demonstration programs used in this manual as a reference for you when working through the examples.
- Appendix C** **Timing Comparisons** lists timing comparisons between 68030 processors and the HP 64430 Emulator. It also provides the DC electrical specifications for the HP 64430 Emulator.

---

## Understanding The Examples

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

If you are not using "pmon", you will find the User Interface/HP-UX Cross Reference appendix of the *68030 Emulation Reference Manual* especially useful. The cross reference table shows you how the "pmon" softkeys translate into commands that can be entered from the HP-UX prompt.

The examples provided throughout this manual use the following structure:

**PRESS** **edit** module.S

**PRESS** or press means you should enter a command by selecting the softkeys and/or typing in any file names or other variables which are not provided in the softkey selections.

**edit** softkeys will appear in bold type. Usually you will not be prompted to use the ---ETC--- softkey to search for the appropriate softkey template. Three softkey templates are available at the HP 64000 system monitor level.

module.S this is the name of a file which you must type in. Softkeys are not provided for this type of selection since it is variable. However, a softkey prompt such as <FILE> will appear as a softkey selection.

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

---

# Notes



# Contents

---

## 1 Introducing The 68030 Emulator

Overview . . . . .	.1-1
Safety Considerations . . . . .	.1-1
Purpose of the 68030 Emulator . . . . .	.1-2
Emulator Features . . . . .	.1-2
Software Debugging . . . . .	.1-2
Symbols . . . . .	.1-2
Real-Time Operation . . . . .	.1-2
Clock Speed . . . . .	.1-3
Emulation Memory . . . . .	.1-3
Analysis . . . . .	.1-3
Registers . . . . .	.1-4
Single-Step . . . . .	.1-4
Breakpoints . . . . .	.1-4
Reset Support . . . . .	.1-4
Memory Management . . . . .	.1-4
Custom Coprocessors Support . . . . .	.1-5
Function Codes . . . . .	.1-5
Foreground or Background Emulation Monitor . . . . .	.1-5
Out-of-Circuit or In-Circuit Emulation . . . . .	.1-6
Manual Coverage . . . . .	.1-6

## 2 Installing Your Emulator

Overview . . . . .	.2-1
Introduction . . . . .	.2-1
Safety Considerations . . . . .	.2-3
Preinstallation Inspection . . . . .	.2-4
Installing Your Emulation System Hardware . . . . .	.2-5
Installation Instructions . . . . .	.2-5
Turn Off Power . . . . .	.2-6
Remove The Card Cage Cover . . . . .	.2-6
Connect The Emulator Pod Cables To The Emulator Boards . . . . .	.2-7
Install Boards Into The Card Cage . . . . .	.2-8

Secure The Pod Cables . . . . .	2-10
Reinstall Card Cage Access Cover . . . . .	2-10



Installing The Emulation Probe Into The Target System . . . . .	2-10
Install Software . . . . .	2-13
Installing 68030 Emulation Software Updates . . . . .	2-13
Turning On The HP 64120A . . . . .	2-14

### 3 Getting Started

Overview . . . . .	3-1
Introduction . . . . .	3-1
Emulation System Used For Examples . . . . .	3-2
Making A Subdirectory For Your 68030 Project . . . . .	3-2
Initializing And Configuring Your Measurement System . . . . .	3-4
Preparing Your Program Modules . . . . .	3-6
Creating The Absolute File In Your Own Subdirectory . . . . .	3-7
Using The Absolute File In The Demo Directory . . . . .	3-8
Preparing The Emulation System . . . . .	3-8
Accessing The Emulation System . . . . .	3-9
Modifying The Default Emulation Configuration . . . . .	3-9
Loading Emulation Memory . . . . .	3-11
Using The Emulator . . . . .	3-11
Displaying Global Symbols . . . . .	3-12
Displaying Local Symbols . . . . .	3-13
Displaying Memory . . . . .	3-14
Modifying Memory . . . . .	3-15
Running from the Transfer Address . . . . .	3-16
Displaying Registers . . . . .	3-17
Using The Step Function . . . . .	3-19
Tracing Processor Activity . . . . .	3-20
Using Software Breakpoints . . . . .	3-24
Using Simulated I/O . . . . .	3-27
Using the DeMMUer . . . . .	3-30
Ending The Emulation Session . . . . .	3-34
Using Command Files . . . . .	3-34



## 4 Answering Emulation Configuration Questions

Overview	4-1
Introduction	4-1
Running Emulation	4-2
Modifying The Configuration File	4-3
Selecting Real-Time/ Nonreal-Time Run Mode	4-4
Enabling Emulator Monitor Functions	4-6
Resetting Into The Monitor	4-8
Enabling Emulator Use of Software Breakpoints	4-12
Selecting The Software Breakpoint Instruction Number	4-12
Defaulting The Stack Pointer For	
The Background Monitor	4-15
Select To Block ECS, OCS Signals	
During Background Monitor Cycles	4-16
Select To Perform Periodic Foreground Accesses	4-17
Selecting Address for Periodic Foreground Access	4-18
Enabling The Foreground Monitor	4-20
Interlock or Provide Termination for	
the Foreground Monitor	4-20
Using Custom Coprocessors	4-21
Specifying The Custom Coprocessor File	4-24
Modifying a Memory Configuration	4-25
Selecting to Block BERR on Non-interlocked	
Emulation Memory	4-28
Mapping Memory	4-28
Memory Map Display Organization.	4-30
Memory Map Definition.	4-31
Emulation Monitor Program	
Memory Requirements.	4-32
Using The Map Command	4-32
Using The Map_overlay Command	4-36
Memory Mapping Example	4-38
Using The Modify Command	4-41
Modify Defined_Codes.	4-41
Modify <ENTRY>.	4-44
Modify Default.	4-46
Deleting Memory Map Entries	4-46
Modify the DeMMUer Configuration	4-46
Ending The Mapping Session	4-48
Modifying The Emulation Pod Configuration	4-49
Configuring for In-circuit Emulation Session	4-50

Enabling DMA Transfers . . . . .	4-52
Enabling DMA Transfers Into Emulation Memory . . . . .	4-52
CPU Clock Rate Determination of Wait States . . . . .	4-53
Disabling On-chip Cache . . . . .	4-54
Enabling MMU For Use During Emulation Session . . . . .	4-54
Modifying Simulated I/O Configuration . . . . .	4-54
Modifying Simulated Interrupt Configuration . . . . .	4-55
Naming The Configuration File . . . . .	4-55

## 5 DeMMUer - What It Is And How It Works

Overview . . . . .	5-1
Introduction . . . . .	5-2
What The DeMMUer Is . . . . .	5-2
How The DeMMUer Operates . . . . .	5-2
When To Use The DeMMUer . . . . .	5-4
When To Turn Off The DeMMUer . . . . .	5-5
Under What Conditions The DeMMUer Will Fail To Work . . . . .	5-5
When To Start The DeMMUer . . . . .	5-6
Startup With The Emulator . . . . .	5-6
The Emulator Was Running Without Using The DeMMUer, And Now I Want To Use It . . . . .	5-6
How To Turn On And Turn Off The DeMMUer . . . . .	5-7
Turn On/Off By Using Emulation Configuration Questions . . . . .	5-7
Turn On/Off By Setting The Analysis Mode . . . . .	5-8
DeMMUer Configuration Setup . . . . .	5-8
How To Access The DeMMUer Configuration Display . . . . .	5-10

## 6 Target System Interface

Overview . . . . .	6-1
68030 Signals . . . . .	6-2
CLK . . . . .	6-2
A(31-0) . . . . .	6-2
FC2-FC0 . . . . .	6-3
R/W . . . . .	6-3
CBREQ . . . . .	6-3
RMC . . . . .	6-3
SIZ0-SIZ1 . . . . .	6-3
CIOUT . . . . .	6-3
AS . . . . .	6-4
DS, DBEN . . . . .	6-4
ECS, OCS . . . . .	6-5

D(31-0)	6-5
DSACK1-DSACK0	6-6
BERR	6-6
HALT, AVEC	6-6
STERM	6-7
CIIN	6-7
CBACK	6-7
BG	6-8
IPEND	6-8
STATUS, REFILL	6-8
BR, BGACK	6-8
IPL2-IPL0	6-8
CDIS, MMUDIS	6-9
RESET	6-9
VCC	6-9
Emulation And Target System DSACK and STERM Signals	6-10
Interlocking Emulation Memory DSACK and STERM and Target DSACK and STERM Signals	6-10
DSACK and STERM Signal Problems In Target Systems	6-11
Use Of Open Collector Drivers	6-11
Early Removal Of DSACK Signals	6-12
Isolating The DSACK Problem	6-12
Using The Vector Base Register	6-13
Using The Internal 68030 Caches	6-14
Cache Control	6-14
Analysis with Cache	6-15
Using Breakpoints With Caches Enabled	6-15
Target Memory Breakpoints	6-16
Emulation Memory Breakpoints	6-16
Using Function Codes For Displaying And Modifying Reserved Address Space	6-17
Enabling/Disabling BERR	6-19
Using DMA	6-19
Using The Run From ... Until Command	6-22
Using The Emulation Foreground Monitor	6-24
Loading the Monitor	6-24
Resetting Into The Monitor	6-24
Memory Access Timing Issues	6-26
33 MHz 68030 Microprocessor	6-26
HP 64430 68030 Emulation System	6-26

Loading An Absolute File . . . . .	6-27
Debugging Plug-in Problems . . . . .	6-29
Review the Configuration . . . . .	6-29
Use the Internal Analyzer . . . . .	6-30
Use the Status Messages . . . . .	6-30
Run Performance Verification (PV) . . . . .	6-31
If All Else Fails . . . . .	6-31

## 7 The Emulation Monitor Programs

Overview . . . . .	7-1
Introduction . . . . .	7-2
Comparison of Foreground and Background Monitors . . . . .	7-3
Background Monitors . . . . .	7-3
Foreground Monitors . . . . .	7-3
Using Both Foreground and Background Monitors in the HP 64430 Emulator . . . . .	7-4
When to Use the Background Monitor . . . . .	7-4
When to Use the Foreground Monitor . . . . .	7-5
Customization of the Monitor Programs . . . . .	7-6
The Break Function And The Emulation Monitor . . . . .	7-6
Emulation Monitor Description . . . . .	7-7
The Exception Vector Table . . . . .	7-7
Emulation Monitor Entry Point Routines . . . . .	7-8
Monitor_entry . . . . .	7-8
Swbk_entry . . . . .	7-8
Jsr_entry . . . . .	7-9
Reset_entry . . . . .	7-9
Exception_entry . . . . .	7-9
Emulation Command Scanner . . . . .	7-9
Emulation Command Execution Modules . . . . .	7-10
Are_you_there . . . . .	7-10
Exit_monitor . . . . .	7-10
Synch_start_enable . . . . .	7-10
Copy_memory . . . . .	7-10
Copy_alt_reg . . . . .	7-10
Mon_alt_registers . . . . .	7-10
Simint_enable . . . . .	7-11
Simint_disable . . . . .	7-11
Sim_interrupt . . . . .	7-11
Customizing The Emulation Monitor . . . . .	7-12
Modifying The Exception Vector Table. . . . .	7-14

Continuing Target System Interrupts	
While In The Emulation Monitor	7-18
Sending Messages From the User	
Program To the Emulator Display	7-18
Emulation Monitor Memory Requirements For The 68030	7-21
Linking The Emulation Foreground Monitor	7-22
Loading The Emulation Monitor	7-22
Using Reset Into Foreground Monitor	7-23

## 8 Using Custom Coprocessors

Overview	8-1
Introduction	8-1
The Custom Register Format File	8-3
Address Specification	8-4
Size Specification	8-4
Name Specification	8-4
Register Set Display Specification	8-5
Emulation Monitor Changes	8-9
Defining a Coprocessor Register Buffer	8-9
Modifying The MON_CPU_REG-ISTERS Table	8-10
Modifying The MON_ALT_REGISTERS Table	8-11
Writing Coprocessor Copy Routines	8-11
Answering Emulation Coprocessor Configuration Questions	8-13

## 9 Using Simulated I/O And Simulated Interrupts

Overview	9-1
Configuring Simulated I/O	9-2
Restrictions On Simulated I/O	9-4
Simulated Interrupts	9-5
How Does A Simulated Interrupt Function?	9-5
Simulated Interrupts Versus Real Interrupts	9-6
Simulated Interrupt Configuration	9-8
Restrictions On Simulated Interrupts	9-10
Modifying The Monitor To Use Simulated Interrupts	9-11

## 10 How The Emulator Works

Overview	10-1
Introduction	10-2
Are You There Function?	10-3
The Run Command	10-4
Run From Command	10-4
Run Until Command	10-5
Run From ... Until Command	10-5
Software Breakpoints	10-7
Setting A Software Breakpoint	10-7
Executing A Software Breakpoint	10-8
Executing A Run Command After	
Executing A Software Breakpoint	10-8
"run from ADDR"	10-9
Single Stepping With Foreground Monitor	10-10
Single Stepping With Background Monitor	10-11
Target Memory Transfers	10-12
Displaying Target Memory	10-15
Copying from Target System Memory	10-16
Modifying Target Memory	10-16
Copying to Target System Memory	10-17
Displaying The CPU Registers	10-18
Modifying The CPU Registers	10-19

### A Emulation Error Messages

68030 Emulation Error Messages	A-1
Attempt to read guarded memory, addr = XXXX	A-1
Attempt to write guarded memory, addr = XXXX	A-1
cannot break into monitor	A-1
Could not disable breakpoint at address XXXX	A-2
Could not enable breakpoint at address XXXX	A-3
monitor did not respond to exit request	A-3
No breakpoint exists at address XXXX	A-4
(no termination) message in tracelist	A-4
no memory cycles	A-4
Reset (with capital "R")	A-4
reset (with lower case "r")	A-5
running	A-5
running in monitor	A-5
slow dev at a= XXXX (YY)	A-5

## B Source Files For Getting Started Examples

Introduction . . . . .	B-1
Source File For towers.c . . . . .	B-2
Source File For simint.c . . . . .	B-9

## C Timing Comparisons

# Illustrations

---

Figure 2-1. Instrumentation Cardcage Features . . . . .	2-2
Figure 2-2. Removing the Cardcage Access Cover . . . . .	2-6
Figure 2-3. ABG Protective Plastic Cable Cover . . . . .	2-8
Figure 2-4. Board Installation Into Cardcage . . . . .	2-9
Figure 2-5. Installing Emulation Probe Into PGA Socket . . . . .	2-12

Figure 3-1. Demonstration Configuration File . . . . .	3-10
--	------

Figure 4-1. Restrict To Real-Time Runs Display . . . . .	4-5
Figure 4-2. Enable Emulation Monitor Display . . . . .	4-7
Figure 4-3. Reset Into Monitor Display . . . . .	4-9
Figure 4-4. Enable Emulator Use of INT7 Display . . . . .	4-10
Figure 4-5. Enable User IPEND Display . . . . .	4-11
Figure 4-6. Software Breakpoint Instruct No. Display . . . . .	4-13
Figure 4-7. Default Stk Pointer for Background Display . . . . .	4-14
Figure 4-8. Block ECS, OCS During Background Monitor . . . . .	4-15
Figure 4-9. Foreground Accesses in Monitor Display . . . . .	4-16
Figure 4-10. Address for Foreground Access Display . . . . .	4-17
Figure 4-11. Enable Foreground Monitor Display . . . . .	4-18
Figure 4-12. Interlock or Terminate Foreground Display . . . . .	4-19
Figure 4-13. Any Custom Registers Display . . . . .	4-21
Figure 4-14. Name of Custom Reg. Format File Display . . . . .	4-23
Figure 4-15. Block BERR on Non-interlock Em Mem Display . . . . .	4-25
Figure 4-16. Default Memory Map Display . . . . .	4-27
Figure 4-17. Overlay Addressing Within Physical Blocks . . . . .	4-33
Figure 4-18. Hexadecimal Address Bit Definition . . . . .	4-33
Figure 4-19. Sample Overlay Mapping #1 . . . . .	4-35
Figure 4-20. Sample Overlay Mapping #2 . . . . .	4-36
Figure 4-21. Modify DeMMUer Configuration Display . . . . .	4-42

Figure 4-22. Configuring for In-circuit Emul. Display . . . . .	4-44
Figure 4-23. Enable DMA Transfers Display . . . . .	4-45
Figure 4-24. CPU Clock Rate Display . . . . .	4-47
Figure 4-25. Disable On-chip Cache Display . . . . .	4-48
Figure 4-26. MMU Enabled During Session Display . . . . .	4-49
Figure 5-1. DeMMUer Configuration Display . . . . .	5-9
Figure 6-1. Memory Access Timing, No DSACK Interlock . . .	6-11
Figure 6-2. DMA Bus Request/Bus Grant Timing . . . . .	6-19
Figure 6-3. DMA Timing Diagram, DMA Disabled . . . . .	6-21
Figure 6-4. Example Stack Frame . . . . .	6-22
Figure 8-1. Sample Custom Register Specification File . . . . .	8-6
Figure 8-2. Custom Reg. Spec. Include File fpu_spec . . . . .	8-7
Figure 8-3. Custom Reg. Spec. File Using Include Files . . . . .	8-8
Figure 9-1. Simulated Interrupt Test Program . . . . .	9-7
Figure 9-2. Simulated Interrupt Function Code . . . . .	9-12
Figure 10-1. Target Memory Transfers in Automatic Mode . .	10-13
Figure 10-2. Monitor Operation At End Of Transfer . . . . .	10-14





# Introducing The 68030 Emulator

---

## Overview

This chapter provides the following information:

- Safety considerations for your emulator
- Purpose of the 68030 emulator
- Features of the 68030 emulator
- What information is given in this manual



---

## Safety Considerations

The HP 64000-UX Microprocessor Development Environment, along with the HP 64430 Emulation Subsystem, is a Class 1 instrument (provided with a protective earth terminal) and meets safety standard IEC 348, "Safety Requirements for Electronic Measuring Apparatus". This Class I instrument meets Hewlett-Packard Safety Class I and has been shipped in a safe condition. Review both the instrument and the manual for safety markings and instructions before operation. Read and become familiar with the "Safety Summary", which follows the Certification/Warranty page of this manual, in addition to the items listed in chapter 2.

---

## Purpose of the 68030 Emulator

The 68030 emulator is designed to replace the 68030 microprocessor in your target system so you can control operation of the microprocessor in your application hardware (usually referred to as the *target system*). The 68030 emulator performs just like the 68030 microprocessor, but is a device that allows you to control the 68030 directly.

---

## Emulator Features

### Software Debugging

The HP 64430 Real-Time Emulator for 68030 microprocessors is a powerful tool for both software and hardware designers. Using the HP 64430 Emulator's emulation memory (up to 2Mega bytes), software debugging can be done without functional target system memory.

### Symbols

Symbolic debugging lets you debug programs using the same symbols that you defined in your source code. You can control program flow using software breakpoints, single-stepping by opcode, and run-from and run-until commands.

### Real-Time Operation

Real-time signifies continuous execution of your program at full rated processor speed without interference from the emulator. (Such interference occurs when the emulator needs to break to the monitor to perform an action you requested, such as displaying target system memory.)

Emulator features performed in real time include: running and analyzer tracing.

Emulator features not performed in real time include: display or modify of target system memory; load/dump of target memory, and display or modification of registers.

## **Clock Speed**

Measurements can be made using the emulator's internal 20 MHz clock or an external clock from 20 to 33.33 MHz with no wait states added to target memory.

## **Emulation Memory**

Memory mapping during an emulation configuration session maps physical memory only. If the MMU is enabled, the user is responsible for knowing user physical memory usage.

Dual-ported memory allows you to display or modify physical emulation memory without halting the processor.

Flexible memory mapping lets you define address ranges over the entire 4 Gbyte address range of the 68030. You can reference emulation memory or target system memory in 256-byte blocks. Blocks can be defined as (1) emulation; RAM or ROM, interlocked, synchronous, asynchronous with a data port width of 8-bits, 16-bits or 32-bits (2) target; RAM or ROM, bus error blocked, cache disabled, burst mode blocked, or (3) guarded access. (Refer to the "Answering Emulation Configuration Questions" chapter for information on memory mapping.)

The 68030 emulator will attempt to break to the emulation monitor upon accessing guarded memory; additionally, you can configure the emulator to break to the emulation monitor upon performing a write to ROM (which will stop a runaway program).

## **Analysis**

The integrated emulation bus analyzer provides real-time analysis of all bus-cycle activity. You can define break conditions based on address and data bus cycle activity. In addition to hardware break, software breakpoints can be used for execution breakpoints. You can select any one of the eight 68030 software breakpoint instructions to be used by the emulator.

When the MMU is enabled, analysis data is physical addresses only, with no symbols. When the deMMUer is enabled, the analyzer can see logical addresses and can display symbols.

Analysis functions include trigger, storage, count, and context directives. The analyzer can capture up to 2047 events, including all address, data, and status lines.

Commands for the HP 64430 emulator and HP 64404A and HP 64405A integrated analyzers have been integrated into one

softkey package, making it easy to make both emulation and analysis measurements.

## **Registers**

You can display or modify the 68030 internal CPU register contents. This includes the ability to modify the program counter (PC) value so you can control where the emulator starts a program run. You can also display or modify the 68030 MMU register contents.

## **Single-Step**

You can direct the emulation processor to execute a single instruction or a specified number of instructions. (If a foreground monitor is selected, the target system trace vector must point to `MONITOR_ENTRY` in the foreground monitor code for single step to function properly). Refer to the "Single Stepping with Foreground Monitor" and "Single Stepping with Background Monitor" paragraphs in chapter 10 for further information.)

## **Breakpoints**

You can set the emulator/analyzer interaction so the emulator will break to the monitor program when the analyzer finds a specific state or states, allowing you to perform post-mortem analysis of the program execution. You can also set software breakpoints in your program. With the 68030 emulator, setting a software breakpoint inserts a 68030 `BKPT` instruction into your program at the desired location.

## **Reset Support**

The emulator can be reset from the emulation system under your control; or your target system can reset the emulation processor.

## **Memory Management**

Memory can be accessed either logically or physically, depending on whether the emulator `deMMUer` is configured to be active or inactive. The on-chip Memory Management Unit (MMU) of the 68030 translates logical (virtual) addresses to physical addresses that are placed on the processor address bus. The `deMMUer` hardware filters the physical address bus to the analyzer. When the `deMMUer` is disabled, it passes the data through unchanged (physical). Symbols, which are in logical memory, are not meaningful when the `deMMUer` is disabled. If the `deMMUer` is configured with MMU information and some ranges of interest, it can track table walks. Tracking the table walks allows the `deMMUer` to maintain a cache of physical to logical translations.

By filtering the physical trace data and substituting logical addresses, the analyzer can then show this logical data with symbols.



## Custom Coprocessors Support

The 68030 emulator does not contain an on-board floating point processor and does not provide support for custom coprocessors in the background monitor mode. It does, however, support custom coprocessors when operating in the foreground monitor mode. In foreground monitor mode, the custom coprocessor instructions can be disassembled in trace displays. You can also display and modify the custom coprocessor registers.

## Function Codes

The HP 64430 emulator supports the use of 68030 function codes. Emulation memory can be mapped to any of the functional address spaces (CPU, supervisor or user, program or data, or undefined). Function codes can be used as an additional specification when referencing memory.




## Foreground or Background Emulation Monitor

The 68030 emulator is supplied with both a foreground and a background monitor. This allows you to trade off between:

- Not using the target system resources but having full logical/physical support with the background monitor.
- Having full interrupt handling and custom coprocessor support with the foreground monitor.

The emulation monitor is a program that is executed by the emulation processor. It allows the emulation controller to access target system resources. For example, when you display target system memory, it is the monitor program that executes 68030 instructions which read the target memory locations and send their contents to the emulation controller.



The monitor program can execute in *foreground*, the mode in which the emulator operates as would the target processor. The foreground monitor occupies processor address space and executes as if it were part of the target program.

The monitor program can also execute in *background*, the emulator mode in which foreground operation is suspended so that emulation processor can be used to access target system resources. The background monitor does not occupy processor address space.

## Out-of-Circuit or In-Circuit Emulation

The HP 64430 emulator can be used for both out-of-circuit emulation and in-circuit emulation. The emulation can be used in multiple emulation systems using other HP 64000-UX Microprocessor Development Environment emulators.

---

## Manual Coverage

This manual provides detailed information on operating the HP 64430 emulator for the 68030 processor. The information in this manual gives 68030 processor specific information. The *68030 Emulation Reference Manual* provides additional information about using 32-bit emulation, including detailed syntactical descriptions of the emulation commands. Detailed operating information for the HP 64404 and HP 64405 integrated analyzers is given in the *Analysis Reference Manual for 32-Bit Microprocessors* and the *68030 Analysis Specifics* manual.



# Installing Your Emulator

---

## Overview

This chapter:

- Reviews the safety considerations for installation.
- Provides preinstallation inspection instructions.
- Shows you how to configure boards in the HP 64120A Instrumentation Cardcage.
- Shows you how to install the emulation system hardware.
- Shows you how to connect the emulation probe cable to your target system.
- Shows you how to turn on the HP 64120A Instrumentation Cardcage.



---

## Introduction

If you are installing your HP 64000-UX components as a new installation, refer to the HP 64000-UX Installation and Configuration Manual for instructions concerning the installation of the HP 64120A Instrumentation Cardcage. Also, refer to the preinstallation instructions given in this section. After you have done these, install the emulation system as instructed later in this section.


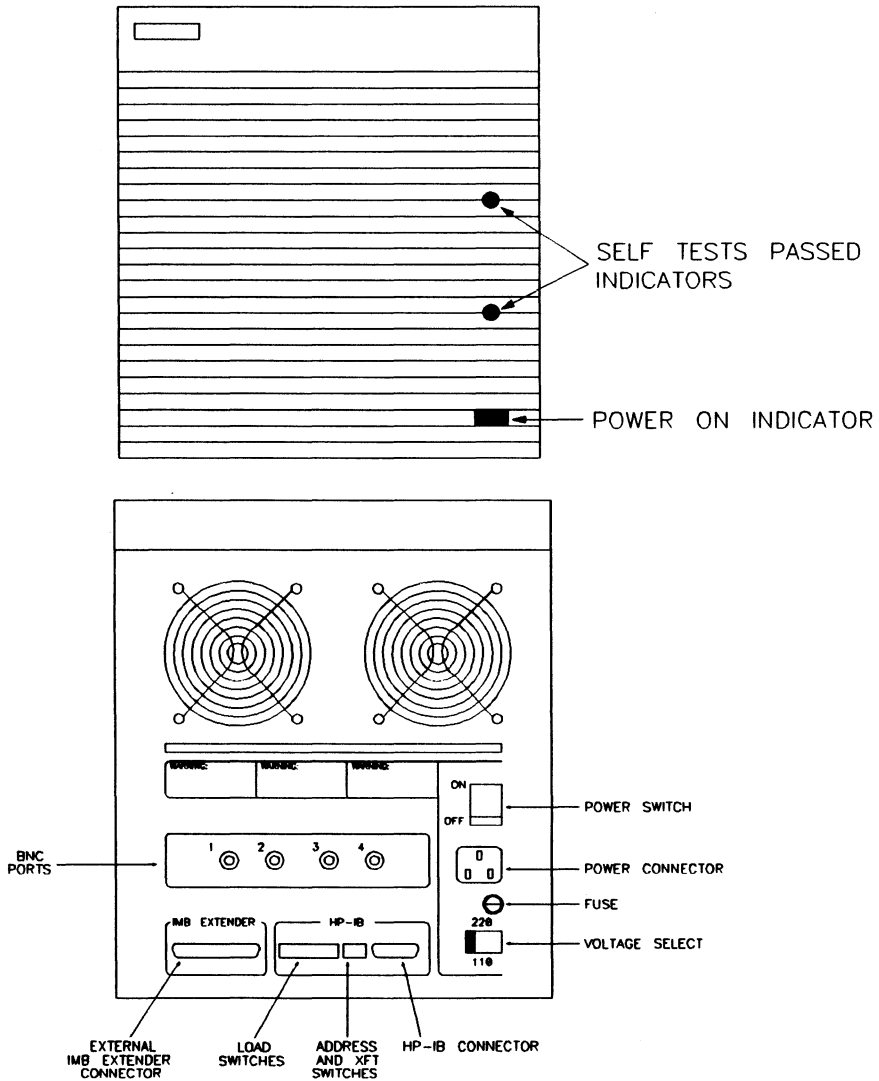


Figure 2-1 identifies some key features of the HP 64120A Instrumentation Cardcage. The identifying labels used in this figure are used throughout this manual. Note the location of the

power switch. For more information on the hardware configuration, refer to the *Installation and Configuration Manual*.



**Figure 2-1. Instrumentation Cardcage Features**



---

## Safety Considerations

The HP 64000-UX Microprocessor Development Environment along with the HP 64430 Emulation System is a Class 1 instrument (provided with a protective earth terminal) and meets safety standard IEC 348, "Safety Requirements for Electronic Measuring Apparatus". This Class I instrument also meets Hewlett-Packard Safety Class I requirements and has been shipped in a safe condition.

The user should review both the instrument and manual for safety markings and instructions before operation. Read and become familiar with the "Safety Summary", printed following the Certification/Warranty page of this manual, and the additional items listed below.

### Warning



---

#### **SHOCK HAZARD! DO NOT ATTEMPT TO DISRUPT PROTECTIVE GROUND!**

Any interruption of the power cord protective conductor (third prong of power cord plug) inside or outside the HP 64120A Instrumentation Cardcage or disconnection of the protective earth terminal in the power source (wall outlet) is likely to make the HP 64000-UX Microprocessor Development Environment **DANGEROUS!** Intentional interruption of the power cord protective conductor is prohibited.

---

### Warning



---

#### **SHOCK HAZARD! ONLY QUALIFIED PERSONNEL SHOULD SERVICE.**

Any adjustment, maintenance, or repair of the opened instrument must **ONLY** be carried out by **QUALIFIED PERSONNEL** aware of the **HAZARDS** involved.

---

## Warning



---

**SHOCK HAZARD! DO NOT USE IF SAFETY FEATURES HAVE BEEN IMPAIRED.**

If the safety features of the instrument have been damaged or defeated, the instrument shall not be used until repairs are made which restore the safety features. The safety features of the instrument could be disabled in the following instances:

1. The instrument shows visible damage.
  2. The instrument fails to perform correct measurements.
  3. The instrument has been shipped or stored under unfavorable environmental conditions. Refer to the Service Supplement portion of this manual for information on the environmental specifications of storage and shipment.
- 

---

## Preinstallation Inspection

Unpack all of the emulation system circuit boards, cables, pod, and related equipment. Carefully inspect the equipment for damage that may have occurred during shipping. If any damage is found, please contact your nearest Hewlett-Packard Sales/Service Office as soon as possible.

Verify that all of the items that you ordered have been shipped. If any equipment is missing, please contact your nearest Hewlett-Packard Sales/Service Office as soon as possible.

---

## Installing Your Emulation System Hardware

This section tells you how to install your emulation hardware into the HP 64120A Instrumentation Cardcage.

Warning



---

**SHOCK HAZARD! INSTALLATION SHOULD ONLY BE PERFORMED BY QUALIFIED PERSONNEL.**

Any installation, servicing, adjustment, maintenance, or repair of this product must be performed only by qualified personnel. Make sure power is off prior to performing any of the installation instructions given below.

---

### Installation Instructions

Proceed as follows to install the Emulation System and related equipment:

Warning



---

**SHOCK HAZARD! HAVE YOU READ THE SAFETY SUMMARY?**

Read the safety summary at the front of this manual before installation or removal of the Emulation Subsystem.

---

Caution



---

**DAMAGE TO CARDS AND CAGE!**

Power to the HP 64120A Instrumentation Cardcage must be removed before installation or removal of option cards (emulation, etc.) to avoid damage to the option cards and the development environment.

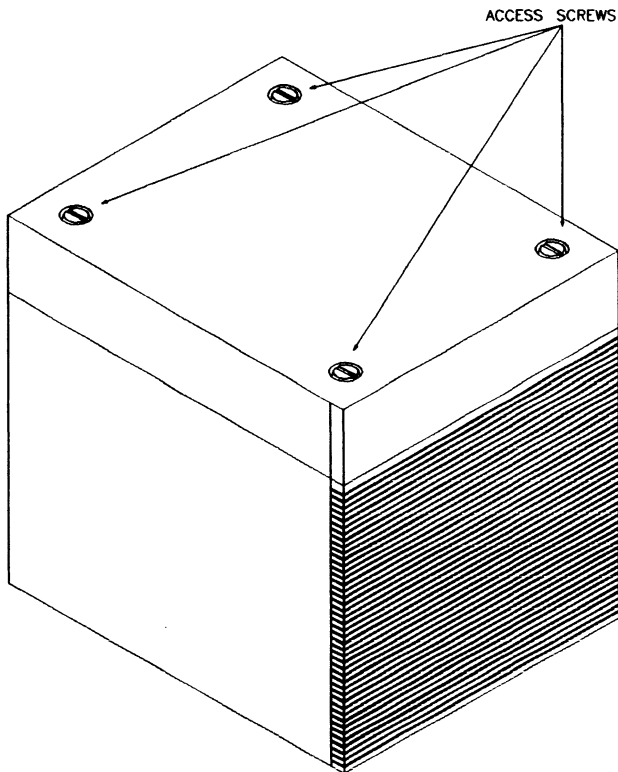
---

### Turn Off Power

Turn OFF power to the HP 64120A Instrumentation Cardcage (see figure 2-1 for the location of the power switch on the HP 64120A Instrumentation Cardcage).

### Remove The Card Cage Cover

The HP 64120A Instrumentation Cardcage access cover is held in place by four screws on the top of the instrumentation cardcage as seen in figure 2-2. Loosen the four screws, and remove the access cover.



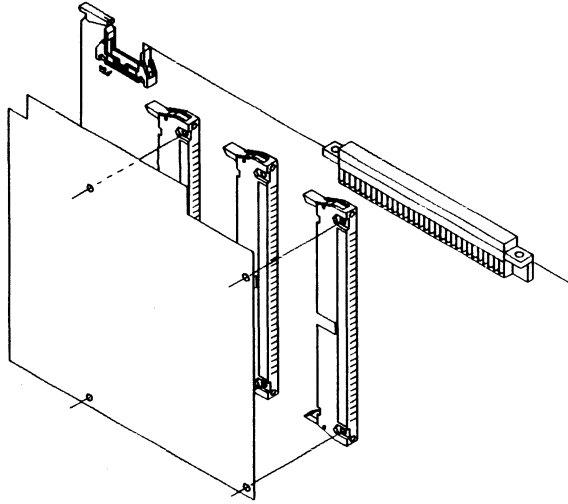
**Figure 2-2. Removing the Cardcage Access Cover**

## **Connect The Emulator Pod Cables To The Emulator Boards**

There are six cables from the emulation pod that must be connected to various cards in the card cage. Connect these cables as follows:

1. Connect the two 44-conductor cables from the pod to the Emulator Control Board (HP 64430-66512). There are no color dots to follow because it does not matter which of the 44-conductor cables are connected to each of the 44-pin connectors.
2. Connect the 50-conductor cable from the pod to the Emulator Control Board (HP 64430-66512).
3. If you are not using the DeMMUer board, connect the three 64-conductor cables from the pod to the Analysis Bus Generator (ABG) board (HP 64411-66503) following the yellow, red, and brown color dots for proper connections.
4. If you are using the DeMMUer board, connect the three 64-conductor cables from the pod to the DeMMUer board (HP 64431-66501) following the yellow, red, and brown color dots for proper connections.

The pod cables connected to the ABG board (64411A) or the DeMMUer board (64431A) are protected by a plastic cover. After connecting the three 64 position cables to the applicable board, secure the plastic cable cover to the board by connecting four screws as shown in figure 2-3. Use a Torx TX 6 screwdriver.



**Figure 2-3. ABG Protective Plastic Cable Cover**

### **Install Boards Into The Card Cage**

Installation of the circuit boards is accomplished by sliding each circuit board into the circuit board guide slots. As you face the front of the HP 64120A Instrumentation Cardcage, the component side of the boards should face the right side of the instrumentation cardcage. Align the connector at the bottom of the board with the motherboard connector at the bottom of the card cage, then apply a downward pressure until the board is seated in the motherboard connector. Be sure the ejector handles are in their full horizontal position when the board has reached its full downward travel.

### **Caution**



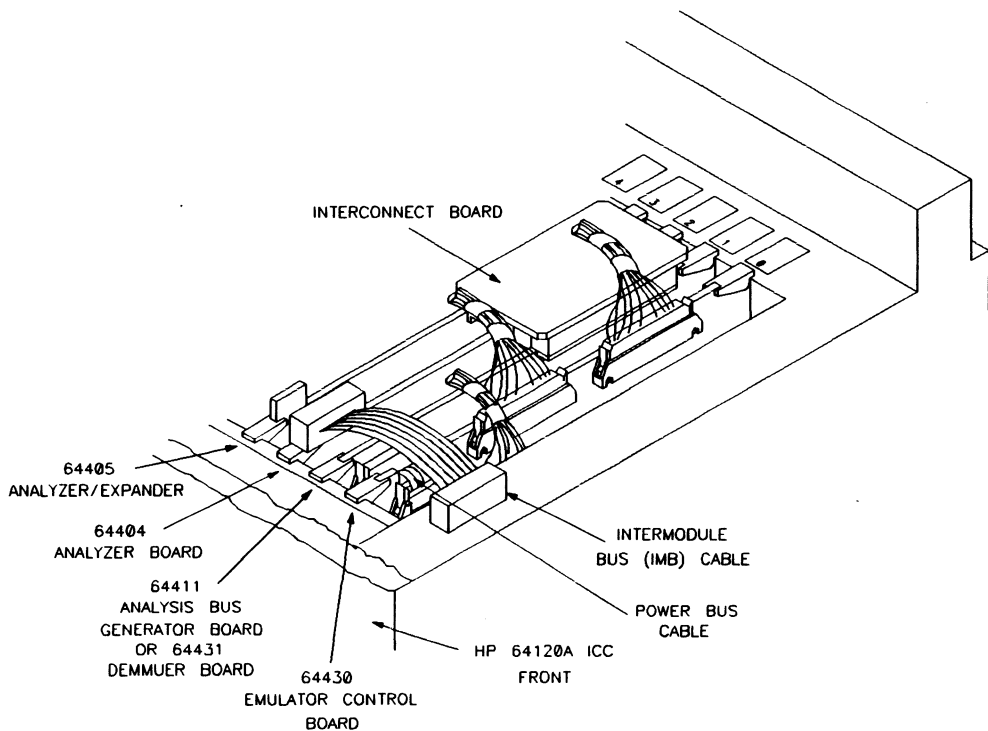
---

**POSSIBLE CABLE DAMAGE!** Be careful to avoid scraping the cables or individual wires with the backs of the printed circuit boards. This will strip insulation from the cables and cause short circuits.

---

Four adjacent card cage slots are required for the circuit boards.  
Install the boards as follows:

1. Install the boards in the card cage in the order shown in figure 2-4.
2. Install the Interconnect Board across the three analysis boards as shown in figure 2-4.
3. Install the power bus cable between the top left edges of the deMMUer board or the analysis bus generator and the emulator control board. This bus is not essential, but will improve reliability of the emulator/analyzer system.



**Figure 2-4. Board Installation Into Cardcage**

## Secure The Pod Cables

Each pod cable has a metal ferrule for strain relief. Snap the ferrule into one of the cable clamps on the instrumentation cardcage. If your instrumentation cardcage does not have cable clamps, you can order them from Hewlett-Packard Co.

## Reinstall Card Cage Access Cover

Reinstall the card cage access cover and secure in place with the hold-down screws.

---

## Installing The Emulation Probe Into The Target System

Caution



---

### **POSSIBLE DAMAGE TO EMULATION PROBE!**

**PROTECT AGAINST STATIC DISCHARGE!** The emulation probe contains devices that are susceptible to damage by static discharge. Therefore, precautionary measures should be taken before handling the microprocessor connector attached to the end of the cable from the emulation probe to avoid damaging the internal components of the probe by static electricity.

---



**Caution**



---

**POSSIBLE DAMAGE TO EMULATION POD!** Do not install the emulation probe into the processor socket with power applied to the target system. The pod may be damaged if power is not removed before installation.

When installing the emulation probe, be sure the probe is inserted into the processor socket so that pin A1 of the emulation probe aligns with pin A1 end of the processor socket. Damage to the emulation equipment may result if the probe is incorrectly installed.

---

**Caution**



---

**POSSIBLE DAMAGE TO TARGET SYSTEM!  
PROTECT YOUR CMOS TARGET SYSTEM COMPONENTS!** If your system includes any CMOS components--turn on the HP 64120A Instrumentation Cardcage first, then turn on the target system; likewise, turn off the target system first, then the development environment.

---

The emulation probe is provided with a pin protector that prevents damage to the probe when not in use (see figure 2-4). **DO NOT** use the probe without a pin protector installed. If the emulation probe is being installed on a densely loaded circuit board, there may not be enough room to accommodate the size of the probe. If this occurs, another pin protector may be stacked onto the existing pin protector.

To install the microprocessor connector in a target system with a Pin Grid Array (PGA) socket (see figure 2-5), proceed as follows:

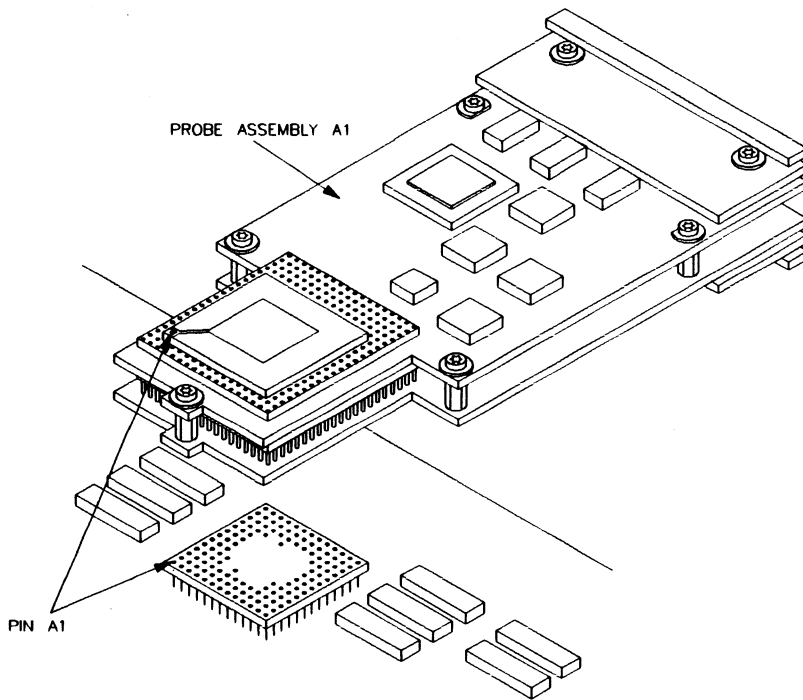
**Caution**



---

**POSSIBLE DAMAGE TO PGA PINS !  
PROTECT PGA PINS FROM DAMAGE!** To avoid damaging the PGA (Pin Grid Array) probe connector pins, use an insertion/extraction tool (such as Augat P/N TX 8136-13) for removing the PGA probe connector.

---



**Figure 2-5. Installing Emulation Probe Into PGA Socket**

1. Remove the 68030 processor from the target system processor PGA socket.
2. Store the 68030 processor in a protected environment. Note the location of pin A1 on both the microprocessor connector and the target system socket.
3. Install the active probe into the target system processor socket.

---

## Install Software

Refer to the Installation Notice that you received with your HP 64000-UX media for complete software installation instructions.

---

## Installing 68030 Emulation Software Updates

After installing a new copy of the 68030 Emulation Software on a system, cycle the power off and then back on for all cardcages containing 68030 emulators. This updates and initializes all emulation software data structures. Run `msinit` before you run your next emulation session. Refer to chapter 3 for a description of `msinit`.

When installing a different revision of the 68030 emulator software, delete all existing ".EB" emulation configuration files. Emulation configuration file names are suffixed by ".EA" and ".EB". The ".EA" file is created when you end out of a "modify configuration" session after giving the configuration a filename. It is an editable file that you can use to modify your configuration without going through the "modify configuration" process during an emulation session. If you do modify it, delete the existing ".EB" file. The ".EB" file is created from an original ".EA" file. It becomes the executable file that the emulation software looks for when you load your emulation configuration file. For this reason you need to delete this file after updating your emulation software, since the new software may have changed something that is in the old ".EB" file. If there is no ".EB" file, the emulation software will use the ".EA" file to build a new one. You need not do anything with the ".EA" file. Questions that are answered in that file but are no longer in the configuration questions are ignored. New questions added to the configuration that are not answered in the ".EA" file are assigned the default answer in the created ".EB" file. You may want to go through the "modify configuration" process and answer all the questions to make sure that your ".EA" file is current after you update your emulation software.

---

## Turning On The HP 64120A

The power switch for the HP 64120A Instrumentation Cardcage is identified in figure 2-1.

### Caution



---

#### **POSSIBLE DAMAGE TO TARGET SYSTEM!**

#### **PROTECT YOUR CMOS TARGET SYSTEM COMPONENTS!**

If your system includes any CMOS components--turn on the HP 64120A Instrumentation Cardcage first, then turn on the target system; likewise, turn off the target system first, then the development environment.

---

Turn the cardcage power on. Three green LED's are visible from the front of the cardcage as seen in figure 2-1. All three should be illuminated to indicate proper operation of the development environment. If all three LED's do not light up, refer to the *HP 64120A Instrumentation Cardcage Service Manual* for information on correcting any problems.



# Getting Started

---

## Overview

This chapter describes how to do the following tasks:

- Create a subdirectory in which you can store your 68030 related files.
- Initialize and define a measurement system.
- Assemble, compile, and link the emulation monitor and demonstration programs by using a makefile.
- Access the emulation system from the monitor level softkeys.
- Modify the default emulation configuration and map memory by loading a configuration file.
- Run an emulation session.

---

## Introduction

This chapter gives an operational overview of the emulation process. The chapter leads you step-by-step through the tasks you must do to prepare your system for emulation and leads you through an emulation session. Emulation features are not explained in depth in this chapter. Its purpose is to familiarize you with the emulation process. Read the entire chapter and go through all exercises in the order presented. This will give you an understanding of the basic operation of the emulator.

---

## Emulation System Used For Examples

The examples given in this chapter (and throughout this manual) were developed with an emulation system including the components listed below.

- HP 64430SX Emulation System (includes Analyzer)
- HP 64874 Cross Assembler/Linker for MC68030
- HP 64907 68030 C Cross Compiler

---

## Making A Subdirectory For Your 68030 Project

Before you start a new project, make a subdirectory for the project. This enables you to keep your files for each project separate from other files. Follow the rules listed below when you make your subdirectory.

- Give the subdirectory a name consisting of from one to fourteen characters. If more than fourteen characters are used, all characters after the fourteenth character are truncated.
- Any characters may be used in the name. Avoid conflict with special characters used in the HP-UX system software by restricting your subdirectory names to alphanumeric characters and the underscore ( `_` ) character.
- Upper and lower case alphabetic characters are significant, i.e., "FILENAME" is a different name than "filename".

## Note



---

The path `/usr/hp64000/bin` must be added to the `PATH` parameter in your `".profile"` file in order to execute HP 64000-UX commands as given in the examples in this manual. Otherwise, you must type the entire path name for HP 64000-UX commands, e.g., `/usr/hp64000/bin/pmon` instead of `pmon`.

---

Do the following steps to make a subdirectory for your 68030 project:

1. Log in to the system using your login and password.
2. Enter **pmon** **Return**. This accesses the HP 64000-UX system monitor. The HP 64000-UX system monitor is softkey driven. You should see softkey labels displayed on your screen.
3. Press the **---ETC---** softkey repetitively until the **makedir** softkey appears as an option on the softkey label line.
4. Press the **makedir** softkey and type in the name you wish to use for your directory (the name **em68030** is used throughout this manual). Press the **Return** key on the keyboard.

**makedir em68030 Return**

You now have a subdirectory named **em68030**.

Whenever you log in to your system to work on the 68030 project, you should change to this directory (using the **chng\_dir** softkey). If you do most of your work on the 68030 project, you can modify your `".profile"` file to change to this directory whenever you log in. If the permissions are set so that you can alter your own `".profile"` file, add the line `"cd $HOME/em68030"` to your `".profile"` file. You will then be in the new subdirectory each time that you log in. If the permissions are set so that you cannot modify your `".profile"` file, see your HP-UX system administrator. The examples in this manual use the **chng\_dir** command to change directories.

---

## Initializing And Configuring Your Measurement System

### Note



---

If you have already initialized the instrumentation cardcage and defined your measurement system, skip this section and go to the next section titled "Preparing Your Program Modules".

Refer to the *Measurement System manual* for the HP 64000-UX Microprocessor Development Environment for detailed information on initializing and configuring measurement systems. The following procedure gives you a brief overview of the initialization and configuration process.

---

To initialize your HP 64120A Instrumentation Cardcage and configure your 68030 emulation system, do the following steps:

1. Press **MEAS\_SYS**.

### Note



---

The **MEAS\_SYS** softkey is displayed after you enter the HP 64000-UX system monitor by executing the **pmon** command.

---

You are now in the `measurement_system` application. The softkeys displayed at this level enable you to initialize and configure your measurement system.

2. Press **msinit Return**.

If you have only one system in your instrumentation cardcage, the softkey label line will disappear and the message "Working" will appear on the STATUS line. After a few seconds, the message "Hit return to continue" will



appear under the STATUS line. Press **Return**. The message will disappear and the softkey labels will return.

If you have more than one system in your instrumentation cardcage, the softkey label line will disappear and the message "Working" will appear on the STATUS line. After a short time, a list of boards in the card cage may be displayed on the screen. Messages may appear on screen asking you to identify the boards in the different systems. After you have identified any boards requested by the system, the message "Hit return to continue" will appear under the STATUS line. Press **Return**. The message will disappear and the softkey labels will return.

3. Press **msconfig Return**.

The screen now displays the module(s) available to be assigned (top of the screen) to a measurement system (middle of the screen).

4. Enter **make\_sys emul683k Return**.

5. Press **add**. If your 68030 emulator is the only system in the instrumentation cardcage, it will be assigned as module 0 as shown at the top of the display. If more than one system is installed in the instrumentation cardcage, the 68030 system module number may be different from 0. Identify the module number of the 68030 emulator shown at the top of the display and type it in from the keyboard. Press **name\_it**, type in **em68030** from the keyboard, and press **Return**. The command line will appear as follows:

```
add 0 naming_it em68030
```

6. Press **end Return**.

This command causes the system to exit the measurement configuration mode and return to the measurement system level.

7. Press **-GOBACK-** to exit the measurement system level and return to the HP 64000-UX system monitor.

The 68030 Emulation module is now defined as module **em68030** in the measurement system (shown in the center of the screen).

---

## Preparing Your Program Modules

Program modules must be assembled or compiled and then linked to create an absolute file. The emulator must be configured with a memory map that allocates memory to the addresses used during execution of the absolute file. Then the absolute file can be loaded into the emulator.

Memory mapping is done for the demonstration programs in this chapter by loading a configuration file supplied with your demonstration software. Memory mapping is described in Chapter 4. The assembly and compile procedures are not described in this manual. Refer to your assembler/linker/librarian and compiler manuals for detailed instructions on these processes.

The next two subparagraphs obtain an absolute file to use for running the getting started procedure. If you have the HP 64874 68030 Assembler/Linker and the HP 64907 68030 C Cross Compiler, create the absolute file by following instructions in the paragraph titled, "Creating The Absolute File In Your Own Subdirectory".

If your system does not have the Assembler/Linker and C Cross Compiler specified above, you can still perform the demonstration emulation procedures in this manual. Follow the instructions in the paragraph titled, "Using The Absolute File In The Demo Directory". A complete absolute file is provided on the media with your emulation software.

Whether you create the absolute file in your own subdirectory or use the absolute file in the demo directory, the absolute file is composed of the following source program modules:

simint.c

Simulated interrupt routines for the demonstration program.

towers.c

The demonstration program. This program solves the popular "Towers of Hanoi" brain teaser puzzle. The program demonstrates many features of the emulator, including simulated I/O and simulated interrupts.

entry.s and os.s

These two programs together set up a virtual system by mapping the 68030 MMU.

Listings of the simint.c and towers.c demonstration programs are included in appendix B of this manual.

## Note



---

The README file in the demo directory contains more information about the demonstration files and their use. To read the README file, enter the following command:

```
!more /usr/hp64000/demo/emul32/hp64430/README
```

---

## Creating The Absolute File In Your Own Subdirectory

The demonstration programs used in this manual are provided on the media shipped with your 68030 emulation system in directory `/usr/hp64000/demo/emul32/hp64430`. Copy these programs to your subdirectory by using the following command:

```
copy /usr/hp64000/demo/emul32/hp64430/* em68030 Return
```

Now enter your subdirectory by using the following command:

```
cd em68030 Return
```

A makefile has been included in the demonstration directory. Use the following commands to have the makefile create the absolute file for the getting started procedure:

```
make clean Return
```

Your display will show files being removed from your directory. These files were built to support the absolute file when it was resident in the demonstration directory.

Now create a complete absolute set of files in your own directory by entering the following command:

**make Return**

During execution of the make file, you will see two "Warning" messages appear on screen. These messages refer to symbols and a variable in the `spmt_demo.c` source file. Both of these Warning messages are normal. They involve a source file that won't be used during the getting started procedures in this manual.

Now go to the paragraph titled "Preparing The Emulation System".

### Using The Absolute File In The Demo Directory

Follow this paragraph if your system does not have the HP compiler and linker needed to create the absolute file for the getting started procedure. An absolute file is supplied in directory `/usr/hp64000/demo/emul32/hp64430`. To run this absolute file, you must change to that directory. To change directories, press the **chn<sub>g</sub>\_dir** softkey and enter the directory pathname `/usr/hp64000/demo/emul32/hp64430`. The command line should appear as follows:

```
cd /usr/hp64000/demo/emul32/hp64430
```

Press the **Return** key. You should now be in the 68030 demo subdirectory. You can verify this by executing the HP-UX **pwd** (present working directory) command.

---

## Preparing The Emulation System

Preparing the emulation system consists of the following steps:

1. Plugging the emulator probe into your target system (for in-circuit emulation), not done in this getting started procedure.
2. Accessing the emulator through the `MEAS_SYS` application.
3. Modifying the default emulation configuration to match your system requirements by loading a configuration file.

4. Loading your absolute file into emulation memory (or target system memory when used).

The following procedures use the emulator in out-of-circuit mode (no target system). Target system plug-in issues are discussed in detail in Chapter 6 of this manual.

## Accessing The Emulation System

Access your emulation system as follows:

1. Press **MEAS\_SYS**.
2. Press **emul683k em68030 Return**.

You are now in the emulation system application. The emulation softkeys are displayed at the bottom of your screen.

## Modifying The Default Emulation Configuration

When you accessed the emulator, the default emulation configuration file supplied with your system was loaded. You need to modify this default emulation configuration to create a configuration that supports your demonstration programs. A configuration file was supplied in the demonstration directory to make these modifications for you. All you need to do is load it.

Load the demonstration configuration file using the command:

**load configuration config Return**

A listing of the demonstration configuration file is shown in figure 3-1. The configuration file sets up the emulation memory map, and answers the emulation configuration questions.

When the emulator is finished loading the memory mapper and background monitor, the STATUS line will show that the emulation processor is Reset. The emulator is ready to be used.

### Note



---

You have two configuration files named "config" in your directory. File config.EB is a binary file used by the emulator. File config.EA is an ASCII file you can edit using an editor on your host system. Emulation configuration files provide an easy method to reconfigure your emulator for desired applications.

---

```
#####
#
# This is the configuration file for the HP 64430 68030 Emulator/Analyzer
# demonstration software.
# If blocks of memory are mapped noncontigously the emulator allocates chunks
# in multiples of 4k bytes.
#####
```

```
BEGIN MEMORY MAP
modify default guarded
modify valid_codes none
### Map 124k memory for all prog and const sections to RAM.
map 00H thru 01efffH emulation ram asynchronous width32
### Map 12k memory for the emulation monitor to RAM.
map 020000H thru 022fffH emulation ram asynchronous width32
### Map 32k memory for the stack.
map 07fff8000H thru 07ffffffH emulation ram asynchronous width32
### Map 88k memory for all data sections.
map 0fffea000H thru 0fffffffH emulation ram asynchronous width32
END MEMORY MAP
```

```
Enable polling for simulated I/O? yes
Function code data space ? none
Simio control address 1? _systemio_buf
Enable polling for simulated interrupts? yes
Function code data space ? none
Simulated interrupt control address? _sim_int_ca
Maximum delay (in milliseconds) for simulated interrupt? 3000
Restrict to real-time runs? no
Enable emulator use of the monitor? yes
Reset into the monitor? yes
Enable emulator use of INT7? yes
Enable user IPEND line during emulator breaks? no
Enable emulator use of software breakpoints? yes
Software BKPT instruction number (0..7)? 7
Default stack pointer for background? 07ffffff8h
Perform periodic foreground accesses while in monitor? no
Address for periodic foreground access? 0
Enable foreground monitor? yes
Interlock or provide termination for foreground? terminate
Any custom registers? no
Name of custom register format file?
Break processor on write to ROM? no
Block BERR on non-interlocked emulation memory? no
In-circuit emulation session? no
Enable DMA transfers? yes
Enable DMA transfers into emulation memory? no
CPU clock rate faster than 25 MHz? no
Disable on-chip cache? yes
MMU enabled during session? no
```

**Figure 3-1. Demonstration Configuration File**

```
Simio control address 2? SIMIO_CA_TWO
Simio control address 3? SIMIO_CA_THREE
Simio control address 4? SIMIO_CA_FOUR
Simio control address 5? SIMIO_CA_FIVE
Simio control address 6? SIMIO_CA_SIX
File used for standard input? /dev/simio/keyboard
File used for standard output? /dev/simio/display
File used for standard error? /dev/simio/display
Block ECS, OCS signals during background monitor cycles? yes
```

**Figure 3-1. Demonstration Configuration File (Cont'd)**

## Loading Emulation Memory

You are ready to begin an emulation session. Before performing emulation, you must load emulation memory with the absolute file created from your source program modules. To load emulation memory, enter the following command:

```
load memory towers Return
```

---

## Using The Emulator

This section demonstrates the use of some of the basic emulator commands. Work through the examples in the sequence given in this section. Otherwise, the displays you get on your workstation screen may not be the same as those shown in the manual. After you have worked through the examples in this section, you may then execute other commands to gain a better understanding of the emulator's operation. See the *68030 Analysis Specifics User's Guide* and the *Reference Manual for 16- and 32 -Bit Internal Analysis* for detailed information on using the emulator's analysis features.

### Note



The displays you obtain on your system for the examples in the following sections of this chapter may vary from those shown in this manual, depending on the type of terminal or workstation you are using.

---

## Displaying Global Symbols

The `display global_symbols` command displays global (externally defined) symbols in the program modules you have loaded into emulation or target memory. To display global symbols, enter the following command:

**display global\_symbols Return.**

You should see a display similar to the following display on your screen.

```
Global symbols in towers
Procedure symbols
Procedure name      Address range      Return  Segment      Offset
__fflush           00005B04- 00005BBB  00005BBA  PROG          000000F2
_bufsync           00005ED2- 00005F0F  00005F0E  PROG          000004C0
_dbl_to_str        00003728- 00003C15  00003C14  COMM          000001E8
_doprnt            00003F3C- 00004F7D  00004F7C  COMM          000000AC
_doscan            00004FD6- 00005291  00005290  PROG          00000000
_exec_funcs        00001E86- 00001EA5  00001EA4  PROG          00000032
_filbuf            000058E8- 00005A11  00005A10  PROG          00000000
_findbuf           00005D86- 00005E27  00005E26  PROG          00000374
_fldbuf           00005BBC- 00005CE9  00005CE8  PROG          000001AA
_memccpy           000061E0- 0000620B  0000620A  PROG          00000000
_startup           000009AE- 00000AE5  00000AE4  PROG          00000000
_wrtchk            00005E28- 00005ED1  00005ED0  PROG          00000416
_xfldbuf           00005CEA- 00005D85  00005D84  PROG          000002D8
atexit             00001E54- 00001E85  00001E84  PROG          00000000
atof               00002B42- 00002BC7  00002BC6  COMM          00000C9A
```

```
STATUS: M68030--Reset_____...R....
display global_symbols
```

```
run      trace      set      step      display  modify  end      ---ETC---
```

You can use the UP and DOWN cursor keys and the NEXT and PREV keys to scroll or page through the global symbols listing.



## Displaying Local Symbols

You can view local symbols within a file or module using the `display local_symbols_in` command. To view local commands in the demonstration program, enter the following command:

`display local_symbols_in towers.c: Return.`

Symbols in towers.c:

Procedure symbols

Procedure name	Address range	Return	Segment	Offset
ask_for_number	0000120C- 0000138B	0000138A	PROG	000000B0
init_display	00001612- 000016C7	000016C6	PROG	000004B6
main	00001162- 00001205	00001204	PROG	00000006
move_disc	00001584- 0000160B	0000160A	PROG	00000428
pause	00001392- 000013C7	000013C6	PROG	00000236
place_disc	00001526- 0000157D	0000157C	PROG	000003CA
remove_disc	000014D2- 0000151F	0000151E	PROG	00000376
show_discs	000013CE- 000014CB	000014CA	PROG	00000272
towers	000016CE- 00001745	00001744	PROG	00000572

Static symbols

Symbol name	Address range	Segment	Offset
S_from	0000000C		
S_reg_param1	00000008		
S_reg_param10	00000014		

STATUS: M68030--Reset\_\_\_\_\_...R....

display local\_symbols\_in towers.c:

run      trace      set      step      display      modify      end      ---ETC---

Note that the ".c" file extension is used to specify C language files and the ".s" file extension is used to specify assembly language files.

## Displaying Memory

The display memory command enables you to view the contents of either emulation or target memory locations. Enter the command:

**display memory main mnemonic Return**

```
Memory      :mnemonic
address     data
1162 4E560000 LINK.W      A6, #0000
1166 2F0B      MOVE.L      A3, -(A7)
1168 2F0A      MOVE.L      A2, -(A7)
116A 45ED800C LEA          ($800C, A5), A2
116E 47FA0222 LEA          ($0222, PC), A3
1172 4EB90000+ JSR          $00000036
1178 42AD8008 CLR.L      ($8008, A5)
117C 60000068 BRA.W      $000011E6
1180 4E71      NOP
1182 48780001 PEA          $00000001
1186 4EB80DEE JSR          $00000DEE
118A 588F      ADDQ.L     #4, A7
118C 42AD8010 CLR.L      ($8010, A5)
1190 42A7      CLR.L      -(A7)
1192 4EBA047E JSR          ($047E, PC)
1196 588F      ADDQ.L     #4, A7

STATUS:  M68030--Reset_____...R....
display memory main mnemonic

run      trace      set      step      display      modify      end      ---ETC--
```

The first address listed in the display is 1162h, the address corresponding to the local symbol **main** in the local symbols display of the towers program. Use the **UP** and **DOWN** cursor keys and the **NEXT** and **PREV** keys to scroll or page through the memory display.

## Modifying Memory

You can modify emulation memory locations mapped as either RAM or ROM. The speed of the towers demonstration program is controlled by the variable `loc_delay`. We will set the value of `loc_delay` to 0 so that the program runs at maximum speed. In order to watch the memory display change as the variable is modified, we will display an area in memory repetitively and then modify the memory. Enter the following command:

**display memory loc\_delay long repetitively Return**

You should see a display similar to the following on your workstation screen.

```
Memory      :long words :blocked :repetitively
address     data        :hex                :ascii
1748-57     000001F4 09095075 7A7A6C65 20776974     .....Pu zzle wit
1758-67     68202564 20646973 63732063 616E2062     h %d dis cs can b
1768-77     6520736F 6C766564 20696E20 2564206D     e solved in %d m
1778-87     6F766573 2E202020 20200A00 0A0A4578     oves. ....Ex
1788-97     65637574 6520276D 6F646966 79206B65     ecute 'm odify ke
1798-A7     79626F61 72645F74 6F5F7369 6D696F27     yboard_t o_simio'
17A8-B7     20746865 6E20656E 74657220 6F6E6520     then en ter one
17B8-C7     6F662074 68652066 6F6C6C6F 77696E67     of the f ollowing
17C8-D7     3A0A094E 756D6265 72206F66 20646973     :..Numbe r of dis
17D8-E7     63732074 6F207573 65205B31 2D25645D     cs to us e [1-%d]
17E8-F7     0A092730 2720746F 20657869 74207072     ..'0' to exit pr
17F8-07     6F677261 6D0A0927 43272074 6F207275     ogram..' C' to ru
1808-17     6E20636F 6E74696E 756F7573 6C792075     n contin uously u
1818-27     73696E67 206C6173 74206E75 6D626572     sing las t number
1828-37     20656E74 65726564 0A0A003F 00256400     entered ...?.%d.
1838-47     20696E76 616C6964 20726570 65617420     invalid repeat

STATUS: M68030--Reset_____...R....
display memory loc_delay long repetitively

run trace set step display modify end ---ETC--
```

Enter the command:

**modify memory long loc\_delay to 0 Return**

Note that the first long word in the display (memory location **loc\_delay**) now shows a long word value of 00000000h.

```
Memory      :long words :blocked :repetitively
address      data          :hex                                     :ascii
1748-57      00000000 09095075 7A7A6C65 20776974      .....Pu zzle wit
1758-67      68202564 20646973 63732063 616E2062      h %d dis cs can b
1768-77      6520736F 6C766564 20696E20 2564206D      e solved in %d m
1778-87      6F766573 2E202020 20200A00 0A0A4578      oves. ....Ex
1788-97      65637574 6520276D 6F646966 79206B65      ecute 'm odify ke
1798-A7      79626F61 72645F74 6F5F7369 6D696F27      yboard_t o_simio'
17A8-B7      20746865 6E20656E 74657220 6F6E6520      then en ter one
17B8-C7      6F662074 68652066 6F6C6C6F 77696E67      of the f ollowing
17C8-D7      3A0A094E 756D6265 72206F66 20646973      :..Numbe r of dis
17D8-E7      63732074 6F207573 65205B31 2D25645D      cs to us e [1-%d]
17E8-F7      0A092730 2720746F 20657869 74207072      ..'0' to exit pr
17F8-07      6F677261 6D0A0927 43272074 6F207275      ogram..' C' to ru
1808-17      6E20636F 6E74696E 756F7573 6C792075      n contin uously u
1818-27      73696E67 206C6173 74206E75 6D626572      sing las t number
1828-37      20656E74 65726564 0A0A003F 00256400      entered ...?.%d.
1838-47      20696E76 616C6964 20726570 65617420      invalid repeat

STATUS: M68030--Reset_____...R....
modify memory long loc_delay to 0

run trace set step display modify end ---ETC--
```

## Running from the Transfer Address

Now that you have used some of the display and modify features of the emulator, it is time to run the demonstration program and use some of the run time features of the emulation system. Enter the following command:

**run from transfer\_address Return**

The STATUS line displays "M68030--Running". This indicates that the demonstration program is executing.

## Displaying Registers

The **display registers** command enables you to look at the contents of the 68030's CPU registers and the contents of the on-board MMU registers. Enter the following command:

**display registers cpu Return**

The contents of the following 68030 CPU registers are displayed on the screen:

program counter (PC)  
source function code register (SFC)  
destination function code register (DFC)  
data registers (D0--D7)  
address registers (A0-A7)

### M68030 Registers

```
NextPC 00000C18      SFC 0 MOT RSVD      DFC 0 MOT RSVD
D0-D7 00000092 00000001 00000092 000058E8 000000FF 00000000 00000064 00000000
A0-A6 000000FF FFFEA057 FFFEA484 FFFEA574 FFFEA054 FFFF21A8 7FFFDF0
USP 7FFFFFF0      1 t o s m i x n z v c      CAAR 00000000
MSP 7FFFFFF0      STATUS 2708 0 0 1 0 7 0 1 0 0 0      VBR 00000000
*ISP 7FFFDE4      wa db e fd ed ibe fi ei
CACR 0000      0 0 0 0 0 0 0 0
```

```
STATUS: M68030--Running_____...R....
display registers cpu
```

```
run      trace      set      step      display      modify      end      ---ETC---
```

user stack pointer (USP)  
 vector base register (VBR)  
 cache address register (CAAR)  
 master stack pointer (MSP)  
 interrupt stack pointer (ISP)  
 status register (STATUS)  
 cache control register (CACR)

Press the **break** softkey, then press **Return**.

The registers display is updated and the status line now reads "STATUS: M68030--Running in monitor". If a display registers command has been executed in the current emulation session, the registers display is updated whenever a break to the emulation monitor program occurs.

#### M68030 Registers

```

NextPC 00000C18      SFC 0 MOT RSVD          DFC 0 MOT RSVD
DO-D7 00000092 00000001 00000092 000058E8 000000FF 00000000 00000064 00000000
AO-A6 000000FF FFEEA057 FFEEA484 FFEEA574 FFEEA054 FFFF21A8 7FFFFDF0
USP 7FFFFFFF0          1 t0 s m i x n z v c      CAAR 00000000
MSP 7FFFFFFF0  STATUS 2708 0 0 1 0 7 0 1 0 0 0      VBR 00000000
*ISP 7FFFFDE4          wa db e fd ed ibe fi ei
                CACR 0000      0 0 0 0 0 0 0 0
  
```

```

NextPC 00000C12      SFC 0 MOT RSVD          DFC 0 MOT RSVD
DO-D7 00000000 00000001 00000092 000058E8 000000FF 00000000 00000064 00000000
AO-A6 000000FF FFEEA057 FFEEA484 FFEEA574 FFEEA054 FFFF21A8 7FFFFDF0
USP 7FFFFFFF0          1 t0 s m i x n z v c      CAAR 00000000
MSP 7FFFFFFF0  STATUS 2704 0 0 1 0 7 0 0 1 0 0      VBR 00000000
*ISP 7FFFFDE4          wa db e fd ed ibe fi ei
                CACR 0000      0 0 0 0 0 0 0 0
  
```

STATUS: M68030--Running\_\_\_\_\_...R....

break

load store copy break reset ---ETC---

## Using The Step Function

The step function enables you to step through your program opcode by opcode. Each time the step command is executed, one program instruction is executed. Enter the command:

**step from transfer\_address Return**

The register display is updated each time a step is executed. In the last entry on the display an additional line is displayed. The address of the instruction executed by the step command and the executed instruction are displayed on the first line of the new register display entry. The step feature is a powerful tool for debugging programs because it enables you to watch the register activity for each executed instruction.

### M68030 Registers

```
NextPC 00000C18      SFC 0 MOT RSVD          DFC 0 MOT RSVD
D0-D7 00000092 00000001 00000092 000058E8 000000FF 00000000 00000064 00000000
A0-A6 000000FF FFFEA057 FFFEA484 FFFEA574 FFFEA054 FFFF21A8 7FFFFDF0
  USP 7FFFFFF0          1 t0 s m i x n z v c      CAAR 00000000
  MSP 7FFFFFF0  STATUS 2708 0 0 1 0 7 0 1 0 0 0      VBR 00000000
*ISP 7FFFFDE4          wa db e fd ed ib e fi ei
          CACR 0000      0 0 0 0 0 0 0 0
```

```
PC 00000400  Opcode LEA          $80000000,A7          4FF98000
NextPC 00000406      SFC 0 MOT RSVD          DFC 0 MOT RSVD
D0-D7 00000092 00000001 00000092 000058E8 000000FF 00000000 00000064 00000000
A0-A6 000000FF FFFEA057 FFFEA484 FFFEA574 FFFEA054 FFFF21A8 7FFFFDF0
  USP 7FFFFFF0          1 t0 s m i x n z v c      CAAR 00000000
  MSP 7FFFFFF0  STATUS 2708 0 0 1 0 7 0 1 0 0 0      VBR 00000000
*ISP 7FFFFFF8          wa db e fd ed ib e fi ei
          CACR 0000      0 0 0 0 0 0 0 0
```

```
STATUS:  M68030--Running_____...R....
step from transfer_address
```

```
run      trace      set      step      display  modify  end      ---ETC---
```

Enter the command:

**step Return**

Note that the emulator executes the instruction stored in the NextPC memory location. Press **Return** repetitively. The emulator executes one instruction each time you press **Return**.

The step instruction enables you to specify a number of steps. This is useful when stepping through program structures such as delay loops. Enter the command:

**step 25 Return**

Notice that the screen is updated with register information each time a program step is executed. While the step command is being executed, the status line displays the message "**MC68030--Steps left #n**" where n is the number of steps remaining. You can use the **NEXT** and **PREV** keys and the **UP** and **DOWN** keys to look at register information that has scrolled off of the screen.

## **Tracing Processor Activity**

The trace function (with analyzer present) enables you to watch each cycle on the processor bus as it occurs. The following examples illustrate some simple uses of the trace function. For more information on the trace function, refer to the *Analysis Reference Manual for 32-Bit Microprocessors* and the *68030 Analysis Specifics User's Guide*.

Enter the command:

**trace TRIGGER\_ON a= long\_aligned main Return**

This sets up a trace of all activity of the bus for 2k bus cycles before and 2k bus cycles after the address labeled main occurs. The **STATUS** line will indicate "Trace in process". Enter the command:

**run from main Return**



After the STATUS line indicates "Trace complete", enter the command:

**display trace Return**

The trace list is displayed on the screen with the trigger state displayed in the center of the screen. Notice the lines prior to the trigger state. The address field shows that these lines represent emulation monitor execution and stack accesses. User program activity is displayed starting with the trigger state (00001160h).

Trace List		Mode:logical data		
Label:	Address		Opcode or Status	time count
Base:	hex		mnemonic	relative
-0007	00000638	\$4E714E73	supr prgm long rd log addr (ds32)	0.12us
-0006	00001FF0	\$41-----	supr prgm byte rd log addr (ds32)	0.16us
-0005	0000063C	\$00000BF6	supr prgm long rd log addr (ds32)	0.08us
-0004	7FFFDE4	\$2704----	supr data word rd log addr (ds32)	0.76us
-0003	7FFFDEA	\$----007C	supr data word rd log addr (ds32)	0.24us
-0002	7FFFDE6	\$----0000	supr data long rd log addr (ds32)	0.24us
-0001	7FFFDE8	\$1162----	supr data word rd log addr (ds32)	0.20us
trigger	00001160	\$4E714E56	supr prgm long rd log addr (ds32)	0.40us
+0001	00001164	\$00002F0B	supr prgm long rd log addr (ds32)	0.24us
+0002	00001168	\$2F0A45ED	supr prgm long rd log addr (ds32)	0.28us
+0003	7FFFDE8	\$7FFFDF0	supr data long wr log addr (ds32)	0.24us
+0004	0000116C	\$800C47FA	supr prgm long rd log addr (ds32)	0.24us
+0005	7FFFDE4	\$FFFEA574	supr data long wr log addr (ds32)	0.24us
+0006	7FFFDE0	\$FFFEA484	supr data long wr log addr (ds32)	0.28us
+0007	00001170	\$02224EB9	supr prgm long rd log addr (ds32)	0.28us
STATUS: M68030--Running		Trace complete _____...R....		
display trace				
run	trace	set	step	display modify end ---ETC--

The address and data values in the default trace list are displayed as hexadecimal numbers. The emulator can also display values in assembly language mnemonics. Enter the command:

**display trace disassemble\_from\_line\_number 0 Return**

Trace List		Mode:logical data	
Label:	Address	Opcode or Status	time count
Base:	hex	mnemonic	relative
trigger	00001160	NOP	0.40us
	= 00001162	LINK.W A6,#\$0000	
+0001			0.24us
	= 00001166	MOVE.L A3,-(A7)	
+0002	00001168	MOVE.L A2,-(A7)	0.28us
	= 0000116A	LEA (\$800C,A5),A2	
+0003	7FFFFDE8	\$7FFFFDF0 supr data long wr log addr (ds32)	0.24us
+0004			0.24us
	= 0000116E	LEA (\$0222,PC),A3	
+0005	7FFFFDE4	\$FFFEA574 supr data long wr log addr (ds32)	0.24us
+0006	7FFFFDE0	\$FFFEA484 supr data long wr log addr (ds32)	0.28us
+0007			0.28us
	= 00001172	JSR \$00000036	
+0008	00001174	\$00000036 supr prgm long rd log addr (ds32)	0.28us
+0009	00000034	NOP	0.36us
STATUS: M68030--Running Trace complete_____...R....			
display trace disassemble_from_line_number 0			
run	trace	set	step
		display	modify
		end	---ETC---

Note that in the updated trace display, the trigger line (line 0) is the first line in the trace display. Address 00001160h corresponds to the **main** label in the demonstration program. The instruction **MOVE.L** on the trigger line is the first instruction in the example program.

You can also display the source program lines corresponding to the traced assembly level code in the trace list. Enter the command:

\* **display trace source on inverse\_video on Return**

```

Trace List                                     Mode:logical data
Label:   Address          Opcode or Status w/ Source Lines      time count
Base:    hex              mnemonic                      relative
trigger 00001160 NOP                               0.40us
        #####towers.c - line   1 thru 128 #####
        static void towers();
        static int  ask_for_number();

        main()
        {
+0001   = 00001162 LINK.W          A6,#$0000                               0.24us
+0002   = 00001166 MOVE.L        A3,-(A7)                               0.28us
        00001168 MOVE.L        A2,-(A7)
+0003   = 0000116A LEA          ($800C,A5),A2
        7FFFFDE8  · $7FFFFDF0    supr data long wr log addr (ds32)  0.24us
+0004   = 0000116E LEA          ($0222,PC),A3                          0.24us

STATUS:  M68030--Running          Trace complete _____...R....
display  trace  source  on  inverse_video  on

run      trace  set      step      display  modify  end      ---ETC--

```

The display is updated with the source code line displayed in inverse video immediately before the related traced assembly level code.

You can use the **UP** and **DOWN** cursor keys or the **NEXT** and **PREV** keys to scroll or page through the entire trace listing. You can copy the trace list to the printer or a file as well.

## Using Software Breakpoints

The `set sw_breakpoints` command lets you set software breakpoints in your program code. This useful feature lets you break execution of your program at the point you select. You can then use the many display and modify commands available in the emulator to examine and debug your code. The emulator replaces the code at the memory location you specify with a 68030 BKPT instruction. You select the appropriate BKPT instruction when answering the emulation configuration questions. Enter the command:

**break Return**

You are now running in the emulator monitor program. Enter the command:

**modify sw\_breakpoints set one\_shot towers.c:ask\_for\_number  
Return**

This command causes the emulator to replace the instruction at the address reference by the symbol `ask_for_number` (0000120CH) with a BKPT 7 instruction. The address specified in the command must be the first address of an opcode. Enter the following command:

**display memory towers.c:ask\_for\_number mnemonic Return**

The display shows a **BKPT 7** instruction at address 0000120CH.

```
Memory      :mnemonic
address     data
120C 484F   BKPT          #7
120E 000048E7 ORI.B       #$E7,D0
1212 3E38242E MOVE.W     $0000242E,D7
1216 0008200D ORI.B       #$0D,A0
121A 0680FFFF+ ADDI.L     #$FFFF82DC,D0
1220 2440    MOVEA.L    D0,A2
1222 47FB8170+ LEA       ($00004FE8,PC),A3
122A 49FB8170+ LEA       ($00005030,PC),A4
1232 2042    MOVEA.L    D2,A0
1234 2610    MOVE.L     (A0),D3
1236 4AAD8008 TST.L     ($8008,A5)
123A 6600013E BNE.W     $0000137A
123E 48780001 PEA       $00000001
1242 4EB80DEE JSR       $00000DEE
1246 588F    ADDQ.L    #4,A7
1248 48780007 PEA       $00000007

STATUS:  M68030--Running          Trace complete _____...R....
display  memory towers.c:ask_for_number mnemonic

run      trace      set      step      display  modify  end      ---ETC--
```

Enter the command:

**run from transfer\_address Return**

The demonstration program runs from the transfer\_address (`_main`) until the BKPT instruction is executed. The BKPT instruction causes the emulator to break into the emulation monitor and the message "STATUS: Software breakpoint hit at address = 120CH" is displayed on the status line.

The emulator's ability to let you set software breakpoints provides you with a method of stopping program execution at a specified point in your program. You can then examine register values,

display or modify memory locations, and perform other operations before continuing execution of your program.

Enter the command:

**display memory towers.c:ask\_for\_number Return**

Note that the instruction **LINK.W** is now displayed at address 120CH in the memory listing. After breaking into the emulation monitor, the emulator replaces the BKPT instruction with the original contents of the memory location (LINK.W instruction).

Memory	:mnemonic		
address	data		
120C	4E560000	LINK.W	A6, # \$0000
1210	48E73E38	MOVEM.L	rm= \$3E38, -(A7)
1214	242E0008	MOVE.L	( \$0008, A6 ), D2
1218	200D	MOVE.L	A5, D0
121A	0680FFFF+	ADDI.L	# \$FFFF82DC, D0
1220	2440	MOVEA.L	D0, A2
1222	47FB8170+	LEA	( \$00004FE8, PC ), A3
122A	49FB8170+	LEA	( \$00005030, PC ), A4
1232	2042	MOVEA.L	D2, A0
1234	2610	MOVE.L	(A0), D3
1236	4AAD8008	TST.L	( \$8008, A5 )
123A	6600013E	BNE.W	\$0000137A
123E	48780001	PEA	\$00000001
1242	4EB80DEE	JSR	\$00000DEE
1246	588F	ADDQ.L	#4, A7
1248	48780007	PEA	\$00000007

STATUS: M68030--Running in monitor Trace complete\_\_\_\_\_...R....  
display memory towers.c:ask\_for\_number

run trace set step display modify end ---ETC--

✕

To continue execution of your program from the point the break occurred, enter the command:

**run Return**

Notice that the status line now reads "**M68030--Running**".

Refer to chapter 10 for a description of how the software breakpoint function is implemented in the 68030 emulator. See chapter 2 of the *68030 Emulation Reference Manual* for the software breakpoint command syntax.

## **Using Simulated I/O**

The demonstration program uses simulated I/O for both entering parameters and displaying the solution to the towers of Hanoi problem. To display the simulated I/O screen, enter the command:

**display simulated\_io Return**

Your screen should appear as shown in the following display.

```
Simulated I/O display           Simulated I/O command: read
display is open                 Return code: 00H

Execute 'modify keyboard_to_simio' then enter one of the following:
Number of discs to use [1-7]
'0' to exit program
'C' to run continuously using last number entered

?

STATUS:  M68030--Running        Trace complete_____...R....
display  simulated_io

run      trace    set      step      display  modify    end      ---ETC---
```

The keyboard must be assigned to simulated I/O before it can be used to specify the number of discs to be used in the program. Enter the command:

**modify keyboard\_to\_simio Return**

The keyboard is now assigned to simulated I/O and is accessible to the demonstration program. Enter the number **7** and press **Return**.



The program then uses simulated I/O to display the solution to the problem on the screen as shown in the following display.

```
Simulated I/O display          Simulated I/O command: write
display is open                Return code: 00H

      ||          ||          ||
      ||          ||          ||
    333||333      ||          ||
    4444||4444    ||          ||
    55555||55555  ||          ||
    666666||666666||          ||
    7777777||7777777||        ||
-----||-----||-----||
      Peg 0          Peg 1          Peg 2

      Solution for Towers with 7 discs.
      Move #3: Move disk 1 from peg 2 to peg 1
      0 simulated interrupts have been serviced.

STATUS:  M68030--Running          Trace complete_____.....

suspend
```

To return control of the keyboard to the host system, press the **suspend** softkey. The normal emulation softkeys will be restored.

For more information on using simulated I/O, see chapter 4 and the *Simulated I/O Operating Manual* supplied with your HP 64000-UX system.

---

## Using the DeMMUer

Use this procedure only if you have a deMMUer in your emulation/analysis system hardware, and you intend to include the functionality of the 68030 MMU in your system. This procedure briefly shows how the deMMUer translates physical addresses which are created by the 68030 MMU into logical addresses for use by the internal analyzer. For more detailed information on DeMMUer operation, refer to the *69030 Internal Analyzer User's Guide*.

The internal analyzer needs logical addresses in order to accept commands you specify using source-file symbols and segment names, and to provide trace lists that show address values using the names of symbols and segments defined in the source file.

Enter the following commands:

```
reset Return  
load configuration virtual Return  
load memory physical os Return
```

The "os" program is a short operating-system script that sets up the 68030 MMU to manage memory for this demonstration program. It allocates 1 megabyte of physical memory for the emulator. It also sets up the deMMUer to match the MMU configuration in "os.s".

In this particular operating system script, physical memory is mapped 1:1 to logical memory. As a result, addresses do not change when the MMU is enabled. Mapping the memory 1:1 is not required for operation of the emulator, but it serves to simplify this example. Enter the following command:

```
set analysis mode logical Return
```

This command turns on the deMMUer so that it will supply logical addresses to the analyzer for storage in its trace memory. Notice that you can select the storage of either **logical** or **physical** addresses with this command. You will find **logical** address information most useful when developing application programs, and physical address information most useful when developing operating systems. The analyzer memory cannot store both logical and physical addresses for each state in trace memory; that is why you make this selection before you start the trace.

Start the analyzer and run the operating system program "os" by entering the following commands:

**trace Return**  
**display trace symbols on Return**  
**run from entry Return**

The "entry" symbol used in the last command is a symbol in the data base of the "os" program. This starts the "os" program running at the appropriate point. Your screen should appear as shown in the following display.

Trace List		Mode:logical address		
Label:	Address	Opcode or Status	time count	
Base:	symbols	mnemonic w/symbols	relative	
-0007				
-0006				
-0005				
-0004				
-0003				
-0002				
-0001				
trigger	AB entry.s:reset	\$00000E2A	supr prgm long rd log addr	-----
+0001	abs 00000004	\$00000A00	supr prgm long rd log addr	0.40us
+0002	ABS STACKTOP	\$2700----	supr data word wr log addr	338.ms
+0003	abs 000F0F02	\$----000F	supr data long wr log addr	12.2us
+0004	abs 000F0F04	\$0000----	supr data word wr log addr	0.60us
+0005	abs 000F0F06	\$----0000	supr data word wr log addr	12.1us
+0006	ABS STACKTOP	\$2700----	supr data word rd log addr	21.4us
+0007	abs 000F0F06	\$----0000	supr data word rd log addr	0.40us
STATUS:	M68030--Running		Trace complete_____	...R....
	run from entry			
run	trace	set	step	display modify end ---ETC--

Enter the following commands:

**break Return**

**load memory towers Return**

The above command causes the towers program to be loaded logically through the monitor. (This requires several minutes.)

**trace TRIGGER\_ON a= long\_aligned main Return**

**run from transfer\_address Return**

The program will run about 15 seconds before the analyzer finds its trigger pointer and captures a trace.

**display trace disassemble\_from\_line\_number 0 Return**

**display trace source on inverse\_video on Return**

In the resulting display that follows, you can see lines of source code (shown in inverse video on your screen), followed by the states in the trace memory that were emitted by those source lines. The trigger line in the display shows the beginning of execution in the towers program. This program will run just like it did before you set out to use memory management. The deMMUer will provide all of the translations required to allow symbols to be used in analyzer commands, and to allow symbolic addresses to be shown in analyzer trace lists.

Trace List		Mode:logical address		
Label:	Address	Opcode or Status		time count
Base:	symbols	mnemonic w/symbols		relative
trigger	000010E8	NOP		0.40us
	#####towers.c - line	1 thru 128	#####	
	static void towers();			
	static int ask_for_number();			
	main()			
	{			
	= 000010EA LINK.W	A6,#\$0000		
+0001	7FFFFFC8	\$00000A5C	supr data long wr log addr (strm)	0.40us
+0002				0.40us
	= 000010EE MOVE.L	A3,-(A7)		
+0003	000010F0	MOVE.L	A2,-(A7)	0.48us
	= 000010F2 LEA	(\$800C,A5),A0		
+0004	7FFFFFC4	\$7FFFFFF0	supr data long wr log addr (strm)	0.40us
+0005				0.40us
STATUS:	M68030--Running	Trace complete	_____	.....
	display trace source on inverse_video on			
run	trace	set	step	display modify end ---ETC---

## Ending The Emulation Session

To end the emulation session, enter the command:

**end release\_system Return**

The system will return to the MEAS\_SYS application level.

This completes your introduction to the 68030 emulation system. You have assembled and compiled program modules, linked your program modules, and used a few of the basic features of the emulation system. For more detailed operational information, refer to the information contained in the other chapters of this manual and the *68030 Emulation Reference Manual*. See the *Analysis Reference Manual for 32\_Bit Microprocessors* and the *68030 Analysis Specifics User's Guide* for detailed information on the analysis features provided in the emulator.

---

## Using Command Files

A command file is a file that lists a series of commands that must be performed to accomplish a particular function. Command files are ideal for setting up, and accessing, the emulation system. Once the file is created, all you need to do is type the file name and press **Return**. The commands in the file will be executed, allowing you to easily enter your emulation session. Refer to the "Creating and Using Command Files" chapter of the *HP 64000-UX User's Guide* for detailed information on command files.



# Answering Emulation Configuration Questions

---

## Overview

This chapter:

- Explains each of the emulation configuration questions.
- Describes how to configure the emulator for compatibility with your 68030 target system.
- Describes how to map your 68030 system memory to emulation and target system memory resources.



---

## Introduction

The 68030 emulator is configured from within the emulation application. When you run emulation for the first time, a default configuration file is loaded. You can modify this file to match your particular system needs by answering a series of emulation configuration questions displayed on your workstation display. After modifying the emulation configuration, you can save it to a file which you can then load each time you enter emulation.

Your answers to the emulation configuration questions define how your 68030 emulator is configured, how resources are shared between the emulator and your target system, how the emulator and target system interact, and what operations are enabled in the emulation environment.

The configuration questions enable you to do the following emulation configuration tasks:

- Selecting real time or nonreal-time run mode.
- Enabling breaks to the emulation monitor.
- Selecting whether to reset into the emulation monitor or to use the user reset exception vector.
- Configuring the foreground and background monitors.
- Enabling and selecting the software breakpoint instruction.
- Configuring custom coprocessor functions.
- Configuring memory.
- Configuring the emulator pod.
- Configuring simulated I/O and interrupts.
- Naming your emulation configuration command file.

---

## Running Emulation

The command sequence to run emulation depends on how you configured your emulation system and what you named it. In this chapter, the example names from chapter 3, Getting Started, are used. To run emulation, do the following steps:

1. Press **MEAS\_SYS**.
2. Press **emul683k em68030 Return**.



---

## Modifying The Configuration File

To modify the configuration, enter the following command:

**modify configuration Return**

A series of questions are displayed on your workstation screen. Your answers to these emulation configuration questions specify the configuration of the emulation hardware and software for a specific application. Each question is displayed with a default response. Additional options are shown in this chapter in parentheses. The default response is selected by pressing the **Return** key. Other responses are selected by pressing the appropriate softkey or by typing in an appropriate response, and then pressing **Return**. If you are modifying an emulation configuration file which you previously made, the default responses are those responses stored in that configuration file.

### Note



---

If you need to return to a question you have already answered, press the **RECALL** softkey. Each time you press **RECALL**, the emulator backs up to the configuration question that was displayed prior to the question currently displayed. You may then make any corrections needed.

---

## Selecting Real-Time/ Nonreal-Time Run Mode

### Caution



---

#### POSSIBLE DAMAGE TO CIRCUITRY!

When the emulator detects a guarded memory access or other illegal condition, or when you execute a command that causes the emulator to break into the emulation monitor, the emulator stops executing the user program and enters the emulation monitor. If you have circuitry in your target system that can be damaged because the emulator is not executing your code, you should use caution. Restrict the emulator to run in real-time mode only. Do not execute commands that cause breaks to the emulation monitor.

---

Real-time refers to the continuous execution of your 68030 program without interference from the development environment except as specified by you. All commands which cause momentary breaks to the emulation monitor are disabled. Momentary breaks are breaks asserted by the emulation software which momentarily diverts 68030 execution to the emulation monitor and then resumes execution of your program. In real-time run mode, you can execute any command which does not cause a break to the emulation monitor. Commands requiring target memory or register accesses are disabled when a user program is running. These commands can only be executed while running in the emulation monitor. An attempt to execute a **run/step from <ADDR>** command while executing the user program in real time causes a break to the emulation monitor.

If the emulator is not restricted to real-time run mode, all selected emulation functions are enabled. Commands requiring access to target memory, logical memory with the MMU, or registers cause a break to the emulation monitor if a user program is running.

All real-time interference can be avoided by disabling the emulation monitor functions. You can select this option later in the configuration questions.

Figure 4-1 shows the display that appears with the following question.

**Restrict to real-time runs? no (yes)**

- no All selected emulator functions are enabled. The emulation system is enabled to break to the emulation monitor whenever a command requiring breaks to the emulation monitor is executed.
- yes Target memory and register accesses are disabled when a user program is running.

```
Not restricted to real time
-----

- All selected emulation functions are available.
- Target memory or register accesses will cause a break/exit.
- Run/step from ADDRESS will cause a break into the monitor.

Restricted to real time
-----

- Target memory or register access, run/step from ADDRESS, etc
  only allowed while the emulator is running in monitor.

STATUS:  Configuring M68030 _____
Restrict to real-time runs? no

yes      no                                     RECALL
```

**Figure 4-1. Restrict To Real-Time Runs Display**

## Enabling Emulator Monitor Functions

The next question asks you if you want to enable emulator use of the monitor. If you answer yes, all emulation commands and features implemented by the emulation monitor are enabled. If you answer no, the next question asked is "Modify memory configuration?" and configuration questions that refer to functions requiring the emulation monitor will not be asked. They will be set to the following default values:

Reset into the monitor?	yes <sup>o</sup> no
Enable emulator use of INT7?	yes no
Software BKPT instruction number (0..7)?	7
Default stack pointer for background?	0ffffffh
Block ECS, OCS signals during background monitor cycles?	yes
Perform periodic foreground accesses while in monitor?	no
Enable foreground monitor?	no

If the emulation monitor is not loaded, all emulation functions that require the monitor for execution will be disabled and their associated softkeys turned off. The functions that require the emulation monitor are:

- automatic reset to monitor
- break
- copy target (logical memory with MMU enabled)
- copy registers
- display target (logical memory with MMU enabled)
- display registers
- emulator use of software breakpoints
- load logical memory
- modify target (logical memory with MMU enabled)
- modify registers
- run from/until <ADDR>
- set break\_on
- step
- store target memory (logical memory with MMU enabled)

Figure 4-2 shows the display that appears with the following question.

**Enable emulator use of the monitor? yes (no)**



## Resetting Into The Monitor

### Note



---

If you answered **no** to the previous question, the following question will not be displayed on your screen.

---

The next question lets you select whether the emulation **reset** command causes the processor to be reset into the emulation monitor or to the memory location specified by the user reset exception vector. This question only affects reset commands entered from the workstation keyboard or processor reset on entry to the emulation module. It has no effect on reset signals generated within the user's target system.

*Does  
not  
effect  
HW reset*

Figure 4-3 shows the display that appears with the following question.

#### **Reset into the monitor? yes (no)**

- yes      The emulation reset command causes the processor to be reset into the emulation monitor. The user-defined reset vector and initial stack pointer are ignored.
  
- no      The emulation reset command causes the processor to fetch the user-defined reset vector and begin execution from that address.

```

Reset to monitor enabled
-----

A reset command, followed by a run command will cause the processor
to enter the foreground monitor if it is enabled and loaded. The
background monitor will be entered automatically after reset if the
foreground monitor is not enabled. The user-defined reset vector will
be ignored.

Reset to monitor disabled
-----

A reset command, followed by a run command, will cause the processor
to fetch the user-defined reset vector and execute from that point.

STATUS:  Configuring M68030_____
Reset into the monitor? yes

yes      no                                     RECALL

```

**Figure 4-3. Reset Into Monitor Display**

Figure 4-4 shows the display that appears with the following question.

**Enable emulator use of INT7? yes (no)**

The emulation break function uses the level 7 interrupt autovector (INT7) processor resource to force the user program to be interrupted and the emulation monitor program to be entered. This question lets you enable or disable the emulation break function, as required for your target system. If your target system cannot share INT7 with the emulator, you need to answer no to this question.

*USE  
CAN  
SHARE STATE  
W/ TARGET  
CANNOT SHARE  
INT7 W/ AV  
IN TRB3*

Emulator use of INT7 enabled

-----  
All selected emulation functions are available.

Emulator use of INT7 disabled

-----  
All emulation break signals are blocked to the processor.  
The following are the only ways to enter the monitor:

- user program jumps to the monitor
- executed exception vector points to the foreground monitor
- a software breakpoint is hit
- reset command with reset-to-monitor function enabled

STATUS: Configuring M68030 \_\_\_\_\_ .....

Enable emulator use of INT7? yes

yes

no

RECALL

**Figure 4-4. Enable Emulator Use of INT7 Display**

- |     |  |
|-----|--|
| yes | All selected emulation functions are available.  |
| no  | All emulation break signals to the processor are disabled.<br>The only ways to enter the monitor program are: <ul style="list-style-type: none"><li>■ user program jumps to the monitor</li><li>■ executed exception vector points to the foreground monitor</li><li>■ software breakpoint is executed</li><li>■ reset command with reset-to-monitor function enabled.</li></ul> |



**Note**



If you answered **no** to the previous question, this question will not be displayed on your screen.

Figure 4-5 shows the display that appears with the following question.

**Enable user IPEND line during emulator breaks?** no (yes)

no

The interrupt pending signal (IPEND) is blocked (driven high) for emulator driven interrupts. Target system generated

```
IPEND during emulation breaks disabled
-----

An emulator-generated break will not send an interrupt pending
signal (IPEND) to the target system. Target system interrupts
will be acknowledged normally.

IPEND during emulation breaks enabled
-----

Any interrupt will send the interrupt pending signal (IPEND) to
the target system.

STATUS: Configuring M68030.....
Enable user IPEND line during emulator breaks? no

yes          no                                     RECALL
```

**Figure 4-5. Enable User IPEND Display**

interrupts cause the IPEND signal to be unblocked (driven low).

yes

Any interrupt sends the interrupt pending signal (IPEND) to the target system.

## Enabling Emulator Use of Software Breakpoints

Software breakpoints must be enabled to allow the language system to pass messages into the background monitor. The next question lets you specify whether or not the emulator can use the 68030 BKPT instructions to do software breaks from the user program into the emulation monitor. The **modify sw\_breakpoints set** and **run until** commands are disabled if you answer **no** to this question. You should answer **no** only if your target system must use all eight 68030 BKPT instructions.

Enable emulator use of software breakpoints? yes (no)

yes

The emulator software breakpoint functions are enabled.

no

Emulator use of software breakpoints is disabled.

## Selecting The Software Breakpoint Instruction Number

The following question lets you specify which of the eight 68030 BKPT instructions the emulator uses to execute software breaks into the emulation monitor.

Note



---

If you answered **no** to the previous question, this question will not be displayed on your screen.

---

Figure 4-6 shows the display that appears with the following question.

Software BKPT instruction number (0..7)? 7 (<number>)

7

Software breakpoint instruction number

-----  
This sets the breakpoint number for the emulation software breakpoint (i.e. BKPT #x, where x is the breakpoint number selected).

STATUS: Configuring M68030 \_\_\_\_\_ .....

Software BKPT instruction number (0..7)? 7

< number >

RECALL

**Figure 4-6. Software Breakpoint Instruct No. Display**

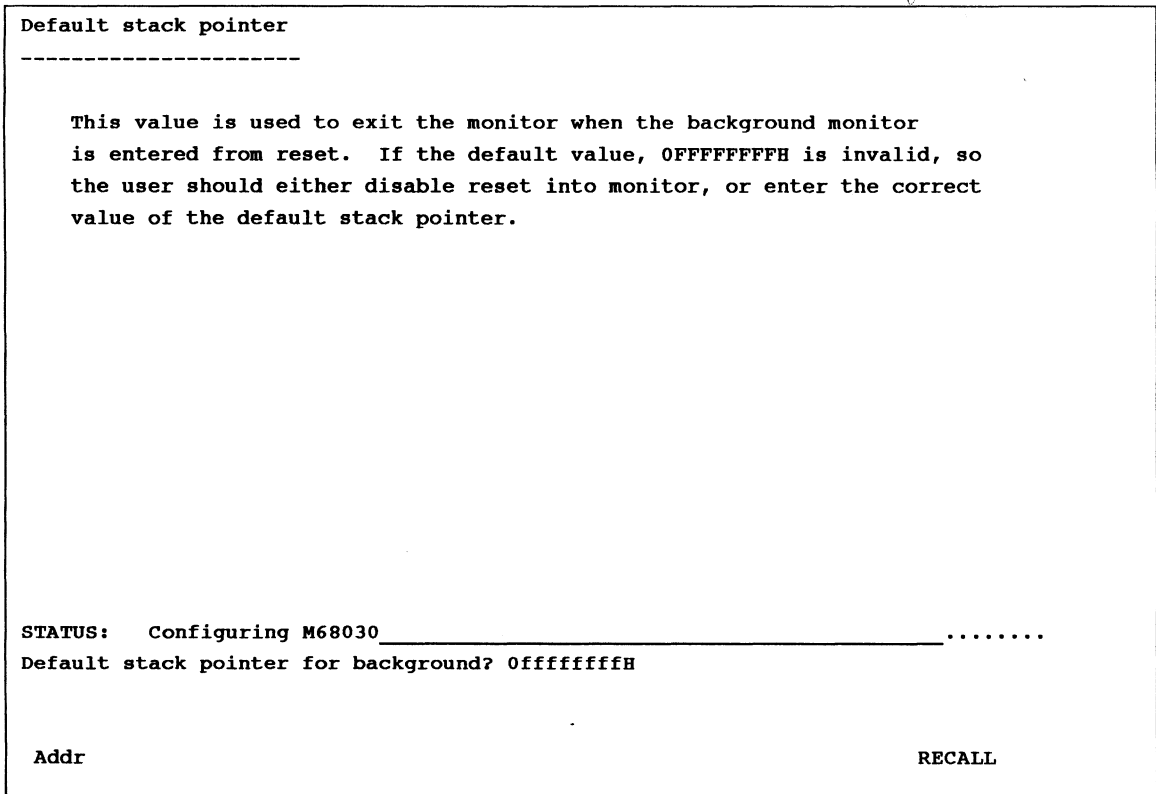
## Defaulting The Stack Pointer For The Background Monitor

This question allows you to set the address value to be used to exit the background monitor if the monitor is entered from reset. The default value (0ffffffh) is not valid so the user must select and enter the correct value in answer to this question, or reset-into-monitor must be disabled.

Figure 4-7 shows the display that appears with the following question.

**Default stack pointer for background? 0ffffffh (<addr>)**

*↑  
Default Stack*



**Figure 4-7. Default Stk Pointer for Background Display**

## Select To Block ECS, OCS Signals During Background Monitor Cycles

Figure 4-8 shows the display that appears with the following question.

Block ECS, OCS signals during background monitor cycles? yes (no)

yes Causes the ECS and OCS signals to be blocked to the target system during background monitor execution.

no Causes the ECS and OCS signals to be unblocked to the target system during background monitor execution. This causes the processor to look as though it is executing out of cache.

```
ECS, OCS signals blocked
-----

During background monitor execution, these signals will be blocked to the
target system.

ECS, OCS signals not blocked
-----

ECS and OCS signals will be visible during background monitor execution.
This will cause the processor to look as though it is executing out of
cache.

STATUS:  Configuring M68030 _____ .....
ECS, OCS signals during background monitor cycles? yes

yes      no                                     RECALL
```

Figure 4-8. Block ECS, OCS During Background Monitor

## Select To Perform Periodic Foreground Accesses

This question determines whether keepalive cycles are generated during background monitor execution. The keepalive function causes a read cycle, at a specified address, to happen periodically. Some target systems need to see continuous cycles. When the background monitor is executing, AS, DS, and DBEN signals are blocked. Figure 4-9 shows the display that appears with the following question.

### Note



If you answer **no** to this question, the next question will be not be asked.

**Perform periodic foreground accesses while in monitor? no (yes)**

Perform periodic foreground accesses while in monitor  
-----

Answering **yes** will cause the background monitor to periodically access an address in foreground space. This is useful for triggering memory refresh strobes in the target, etc.

STATUS: Configuring M68030\_\_\_\_\_ .....

Perform periodic foreground accesses while in monitor no

yes

no

RECALL

**Figure 4-9. Foreground Accesses In Monitor Display**

## Selecting Address for Periodic Foreground Access

Figure 4-10 shows the display that appears with the following question.

### Note



If you answer **no** to the previous question, thus question will be not be asked.

Address for periodic foreground access? 0 (<addr>)

Foreground access address  
-----

This address will be used by the background monitor to access the foreground address space. The access will be a read cycle.

STATUS: Configuring M68030 \_\_\_\_\_

Address for periodic foreground access? 0

Addr RECALL

Figure 4-10. Address for Foreground Access Display

## Enabling The Foreground Monitor

This question allows you to choose whether or not to use the foreground monitor. The foreground monitor may be loaded by the background monitor. If you choose to use the foreground monitor, only the **load memory** command (of all the commands that require the monitor) is allowed. The foreground monitor can operate without disabling any interrupts, and it allows full user defined coprocessor support. However, the foreground monitor takes up target resources, and does not allow physical memory access when the MMU has been configured to be enabled. Figure 4-11 shows the display that appears with the following question. If this question is answered **no**, the sequence will skip to the "Modify memory configuration?" question.

Enable foreground monitor? no (yes)

### Foreground monitor enabled

-----

The user may load a foreground monitor to provide functionality. This monitor may be loaded using the background monitor. Other than the memory access for the load, no commands requiring monitor functionality are allowed until the foreground monitor has been loaded. The foreground monitor can run without disabling interrupts and also allows the emulator to support user-defined coprocessor registers access. A foreground monitor takes up target system resources and will not support physical memory access when the MMU is enabled in configuration.

### Foreground monitor disabled

-----

The background monitor will be used for all monitor functionality. Generic coprocessor register access is not supported and interrupts are blocked during the monitor execution. Both logical and physical memory accesses are supported and target resources are not used.

STATUS: Configuring M68030 \_\_\_\_\_ .....

Enable foreground monitor? no

yes            no

RECALL

Figure 4-11. Enable Foreground Monitor Display



## Interlock or Provide Termination for the Foreground Monitor

This question allows you to determine whether the foreground monitor CPU space cycles will be terminated immediately as an asynchronous cycle or if the cycles will be interlocked with the target system cycles.

Figure 4-12 shows the display that appears with the following question.

**Interlock or provide termination for foreground? terminate (intrlck)**

```
Interlock foreground cycles
-----

Foreground monitor CPU Space cycles will be interlocked with the
target cycles, as determined by the map entry for the foreground
address space.

Emulator terminates foreground cycles
-----

The cycle is terminated immediately as an asynchronous cycle.

STATUS: Configuring M68030_____.....
Interlock or provide termination for foreground? terminate

intrlck   term                                     RECALL
```

**Figure 4-12. Interlock or Terminate Foreground Display**

## Using Custom Coprocessors

### Note



---

Custom register access is only supported with the foreground monitor enabled, except in the case of the MMU registers. MMU registers display and modification are supported for both foreground and background monitors.

---

The 68030 emulator has the capability to access floating point processors, memory management units, and other coprocessors in your target system. You can both display and modify coprocessor register sets. In order to use custom coprocessors with the emulator, you must provide a custom register format file defining the coprocessor register set and modify the emulation monitor program as described in chapter 8 of this manual. This must be done prior to modifying the emulation configuration.

Figure 4-13 shows the display that appears with the following question.

**Any custom registers? no (yes)**

yes

The emulator is enabled to access the custom coprocessors that you have defined in your custom register format file.

no

Use of custom coprocessors is disabled.

Custom coprocessor support

---

This question should be answered "yes" if it is desired to access coprocessor register sets, such as an external FPU. The user is expected to supply monitor code to read and write register contents.

STATUS: Configuring M68030 \_\_\_\_\_

Any custom registers? no

yes

no

RECALL

**Figure 4-13. Any Custom Registers Display**

## Specifying The Custom Coprocessor File

### Note



---

If you answered **yes** to the question "Any custom registers?", the following question will be displayed on your screen.

---

Figure 4-14 shows the display that appears with the following question.

**Name of custom register format file? (<FILE>)**

The default answer to the question is NULL. The MMU is supported internally. There is an example custom register format file provided with your emulation software. The example is located at `/usr/hp64000/inst/emul32/0410/0204/custom_spec`. If you are using custom coprocessors, you must enter the full pathname of the custom register format file that you made for these coprocessors.

```
Custom register format file
-----

This file supplies the format specification for the custom
register sets that are enabled. Information about the register
sets is present including the display format. The MMU is
excepted from needing this formatting data.

STATUS: Configuring M68030_____
Name of custom register format file?

FILE                                     RECALL
```

**Figure 4-14. Name of Custom Reg. Format File Display**

**Modifying a Memory Configuration**

When you begin your initial emulation session you must configure (map) the memory space you will be using. The configuration you need is based on your user program requirements and on the configuration of your target system, if one is available. As you progress with your program development, your memory map requirements will probably change. As your requirements change, you will need to modify your configuration file.

The following questions let you review and modify the memory configuration stored in the emulation configuration file.

## Note



---

If you answer **no** to the "Modify memory configuration?" question, the sequence will skip to the "Modify emulator pod configuration?" question.

---

### Modify memory configuration? no (yes)

- yes      Allows memory mapping and deMMUer configuration to be modified. The current memory map is displayed. Memory configuration is explained in the following sections.
- no      Allows you to skip memory configuration if you do not want to change memory usage. A **no** response causes the memory to be configured as specified by the current emulation configuration file. If **no** is entered, the next question is "Modify emulator pod configuration?"

### Break processor on write to ROM? yes (no)

- yes      A break to the emulation monitor occurs if the processor attempts to write to a memory location mapped as emulation or target ROM.
- no      Breaks are not generated when the processor attempts to write to memory locations mapped as emulation ROM.

If write operations to emulation memory mapped as ROM are attempted during program execution, the contents of emulation memory are not modified. Write operations resulting from emulator commands that modify memory (e.g., load and modify) will modify the contents of emulation memory locations mapped as ROM if the MMU is disabled or it is a physical access.

Write operations to target memory mapped as ROM may or may not alter memory contents, depending on your target system hardware.

## Selecting to Block BERR on Non-interlocked Emulation Memory

Figure 4-15 shows the display that appears with the following question.

**Block BERR on non-interlocked emulation memory? no (yes)**

yes      Bus errors (BERR) that occur during emulation memory cycles (if the address is configured as non-interlocked) are blocked. This allows the monitor or other user program to run in a memory space not usually allowed by the target system hardware. This does not prevent retry operations from happening.

```
BERR disabled
-----

Bus errors (BERR) that occur during emulation memory cycles when
the emulation memory address is configured as non-interlocked
will be blocked. This allows the monitor or other user programs
to run in a memory space not normally allowed by the target
system hardware. The BERR signal is not blocked if it is part
of a retry operation.

BERR enabled
-----

All bus error signals not excluded by a unique mapping are admitted to
the processor.

STATUS:  Configuring M68030_____.....
Block BERR on non-interlocked emulation memory? no

yes      no                                     RECALL
```

**Figure 4-15. Block BERR on Non-interlock Em Mem Display**

no All bus error signals (BERR) are transmitted to the processor.

### Mapping Memory

After you answer the question "Break processor on write to ROM?", the emulation memory map is displayed. The 68030 processor memory space required for your applications must be mapped to emulation memory, target memory, or guarded memory. Emulation memory is memory that is physically located in the emulation pod. Target memory is memory that is physically located in your target system. Memory mapped as guarded is memory that, under normal conditions, should not be accessed by your target system. Any reference to the address space mapped as guarded memory will result in an emulation memory break and the display of the following error message:

```
STATUS: 68030--Running in monitor    Guarded access a= <ADDR> (<FC>)
```

where <FC> is a two letter mnemonic describing the function code of <ADDR>.

The memory mapper must be properly programmed to correspond to emulation memory and target system memory resources in order for emulation to work correctly. The memory mapper allows you to divide the processor's address space into blocks that can be individually configured to have any of the following attributes:

- Emulation memory; RAM or ROM; synchronous; interlocked; or asynchronous with 8-bit, 16-bit, or 32-bit width;
- Target memory; RAM or ROM; bus error blocked; cache disabled; burst mode blocked.
- Guarded memory.



During emulation, the memory mapper monitors the address bus and provides the attributes for the address present at any given time. This information is used by the emulator hardware to control the flow of data between the emulation processor and the memory resources.

**Memory Map Display Organization.** The default memory map display is shown in figure 4-16. Each entry line shows the entry number, address range starting value, address range ending value, function code of the address range, attributes of the entry, and overlay definition. The overlay definition shows the number of the entry being overlaid, and the address in the memory map entry being overlaid that corresponds to the starting address of the overlay entry.

```

Mapping memory:  Function codes = OFF
ENTRY      START      END      ATTRIBUTES      OVERLAY
  1         0H      FFFFFFFFH  TARGET RAM      (CS)

STATUS:  Mapping emulation memory, default mapping:  guarded _____...R....
end

map      map_over  modify  delete      display  deMMUer  end

```

**Figure 4-16. Default Memory Map Display**

Softkey labels are displayed for the commands available in the memory mapper. You can specify individual map entries, overlay existing map entries, modify existing entries (including the default mapping attributes), modify the deMMUer configuration, delete currently defined entries, or end the map definition session. These commands are described in the following sections.

**Memory Map Definition.** The memory map partitions the processor address range into blocks defined as emulation RAM or ROM, target RAM or ROM, or guarded (illegal) space. Each entry defines a particular address range as one of the five possible memory types.

Memory entries can be further defined by function code. Emulation memory can also be assigned cycle type of synchronous, interlocked, or asynchronous with a bit-width of 8, 16, or 32. Based on the cycle type, emulation memory returns the appropriate signals to the 68030 processor.

Any address range not defined by an entry is mapped to the memory default. The addresses entered are physical addresses at the appropriate 68030 pins.

The memory mapper has a resolution of 256 (ffH) bytes. Once the mapper software processes the inputs, the entry range is rounded to integral multiples of 256 bytes. The final range includes all of the specified memory space, plus the remainder of any 256-byte blocks which were partially specified. Any parts of the 68030 address range not defined by an entry are mapped to the memory default.

## Note



---

If the end address of a specified address range is the same as the first address of a 256-byte memory block (e.g. 100h, xxxxxx00h, etc), the end address value is rounded down one byte (e.g. to 0ffh, xxxxxxffh, etc.)

This can cause a problem if you attempt to specify an address range with the same start and end address corresponding to the first address of a 256-byte memory block. If you enter the command:

```
map 100h thru 100h emulation ram Return
```

the error message "**ERROR: Lower address in range greater than upper address**" is displayed. This command is not allowed by the emulator because the ending address (when rounded down to 0ffh) is less than the starting address (100h).

---

All emulation memory is displayed and loaded directly by the emulation software by way of the memory port assigned to the host processor. If the MMU is enabled, logical addresses are done using the emulation monitor. Physical addresses are still done using the host port. Any attempt by the 68030 CPU to write to memory mapped as emulation ROM will not change the contents of that memory location.

When target memory is specified for a given address range, all memory cycles using that address range access the target system. All memory load and display operations for your target system are done via the emulation monitor.

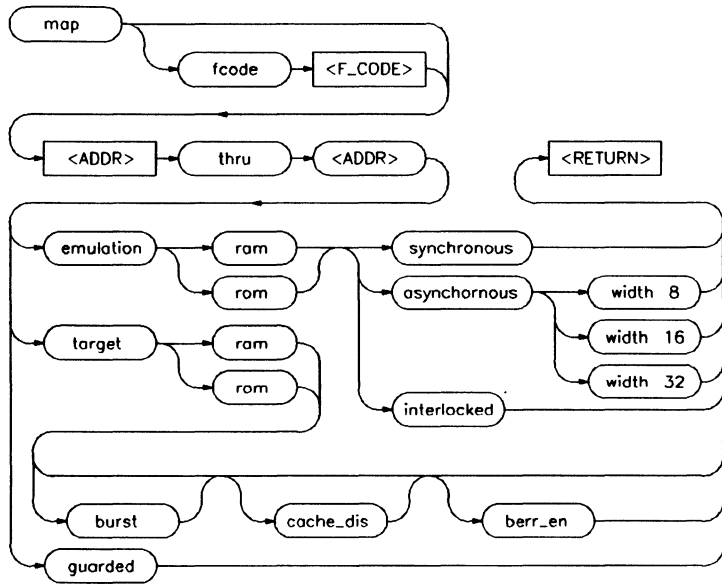
Multiple processor address ranges can be overlaid onto the same physical emulation memory by using the map\_overlay command. Overlaying applies only to emulation memory. The emulator has no control over your target system memory resources.

**Emulation Monitor Program Memory Requirements.** You need to know certain information about the emulation monitor (delivered as part of your emulator software package) prior to linking the monitor program and mapping memory space. Chapter 7 gives a detailed description of the emulation monitor, including memory requirements for the program. Refer to the paragraphs titled "Emulation Monitor Memory Requirements" in chapter 7 for a full description of the emulation monitor memory requirements.

## Using The Map Command

All memory map entries are made up of an address range and attributes which specify the type of memory accessed by the specified address range. In addition, a specific function code and address width (port size) can be assigned to a memory map entry. Memory mapping is done using the map command.

Mapper blocks are entered using the following command syntax:



where:

**fcode** lets you assign a function code to a memory map entry. The function codes enabled for your particular configuration are displayed on softkeys after you press the fcode key. If you specify **modify defined\_codes none**, the fcode attribute is disabled and the softkey is not displayed. You can specify user-defined function codes by typing in the numeric value of the function. See the section in this chapter on the **modify defined\_codes** command for more information on function codes.

<ADDR>	defines a bit pattern of up to 32 bits which specifies a particular location in memory. That bit pattern can be entered as a binary, octal, hexadecimal, or decimal number.
guarded	designates an address range which is not expected to be accessed. Any processor access to a location within such a range will result in a break of the program execution. No emulation memory is used when an address range is specified as <b>guarded</b> .
emulation	designates memory supplied by the emulation system.
rom	designates memory which cannot be modified by the 68030 processor. Emulation memory that is actually RAM but is mapped as ROM performs as ROM during emulation. The host can read and write to ROM.
ram	designates memory which can be read from or written to without restriction.
inter-locked	designates that the memory entry is defined to interlock cycles with your target system.
synchronous	designates that the memory entry is defined for synchronous cycle access.
asynchronous	designates that the memory entry is defined for asynchronous cycle access.
width8	defines the memory map entry to be an 8-bit data port.
width16	defines the memory map entry to be a 16-bit data port.
width32	defines the memory map entry to be a 32-bit data port.
target	designates memory supplied by your target system. Mapping an address range to target space requires no emulation memory.

burst enables the burst mode access.

cache\_dis disables caching for address range.

berr\_en enables bus errors for the entry.

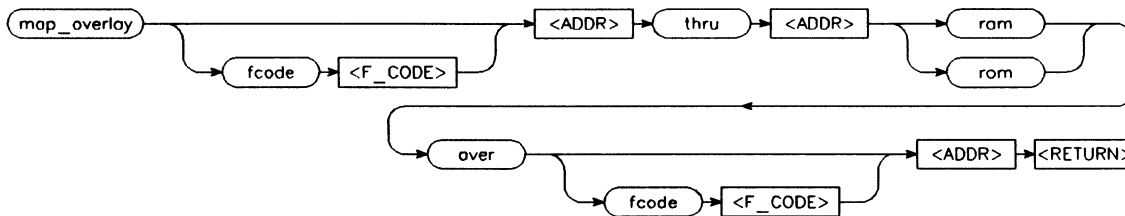
The first <ADDR> of a range specification should be the starting address of a block boundary. If an address inside a memory block area is entered, the system converts this address to the starting address of the block prior to its mapping. Leading zeros may be deleted as long as the most significant digit is numeric.

The minimum map entry size is 256 bytes. The maximum size is the number of available blocks.

### Using The Map\_overlay Command

When making a memory map, you can enter "overlay" addresses in emulation memory hardware blocks. With this feature, you can cause a single block to function as if it were several different blocks, each corresponding to a different set of addresses. Memory overlaying applies only to emulation memory. The emulator has no control over target system resources.

Map overlays are entered using the following command syntax:



Map\_overlay command parameters have the same definitions as those listed for the map command parameters.

There are some restrictions imposed on the map overlay function by the physical structure of emulation memory. Emulation memory is physically made up of 4K byte blocks of memory as shown in



When overlaying memory, the address of the memory overlay and the address of the memory location must be mapped to the same 256 byte block in the 4K byte physical memory block, e.g., the third least significant hexadecimal digit in the specified addresses must be identical. For example, the command:

```
map_overlay fcode SUPER_DATA 0f00f800h thru  
0f00f8ffh rom over fcode SUPER_PROG  
0002800h Return
```

is a valid command. However the command:

```
map_overlay fcode SUPER_DATA 0f00f800h thru  
0f00f8ffh rom over fcode SUPER_PROG 0002a00h  
Return
```

is not a valid command. An attempt to execute the last command would cause the error message "Offset for overlay does not match emulation address" to be displayed.

### **Memory Mapping Example**

The following example shows how to map memory in a system made up of a target system with some memory installed and the 68030 emulator. This example shows how to use the map and map\_overlay commands. Before defining the new memory map, delete all entries in the current map. Enter the following commands:

```
delete all Return  
modify defined_codes all Return
```

The memory map display will appear as shown in figure 4-19. Note that one entry is still displayed. The CPU\_SPACE mapping to target RAM cannot be deleted by the user. This address space is required for vectored exception processing.



```

Mapping memory:  Function codes = ON
ENTRY          START          END          FC          ATTRIBUTES          OVERLAY
  1             0H  FFFFFFFFH  (CS)  TARGET RAM

STATUS:  Mapping emulation memory, default mapping:  guarded _____ ...R....
modify  defined_codes  all

map      map_over  modify  delete          display  deMMUer  end

```

**Figure 4-19. Sample Overlay Mapping #1**

CPU\_SPACE must be mapped to target memory so that vectored exceptions will not interfere with emulation functions.

Type the following entries into the memory map.

**map fcode USER\_DATA 0 thru 0ffffh emulation ram asynchronous width32 Return**

**map fcode USER\_PROG 18000000h thru 1800ffffh emulation rom asynchronous width32 Return**

**map fcode SUPER\_DATA 0 thru 3fffh target rom burst cache\_dis berr\_en Return**

map fcode SUPER\_PROG 0 thru 3ffh target rom burst  
cache\_dis berr\_en Return

map fcode SUPER\_PROG 0f000000h thru 0f000fffh  
emulation ram asynchronous width32 Return

map\_overlay fcode SUPER\_DATA 0f000000h thru 0f000fffh  
ram over fcode SUPER\_PROG 0f000000h Return

The memory map resulting from these commands is shown in  
figure 4-20.

Mapping memory:		Function codes = ON				
ENTRY	START	END	FC	ATTRIBUTES	OVERLAY	
1	0H	FFFFH	(UD)	EMUL RAM [32 bits]		
2	18000000H	1800FFFFH	(UP)	EMUL ROM [32 bits]		
3	0H	3FFH	(SD)	TARGET ROM [burst/berr]		
4	F000000H	F000FFFH	(SD)	EMUL RAM [32 bits]	F000000H (6)	
5	0H	3FFH	(SP)	TARGET ROM [burst/berr]		
6	F000000H	F000FFFH	(SP)	EMUL RAM [32 bits]	F000000H	
7	0H	FFFFFFFH	(CS)	TARGET RAM		

STATUS: Mapping emulation memory, default mapping: guarded\_\_\_\_\_...R....  
map\_overlay fcode SUPER\_DATA 0f000000h thru 0f000fffh ram over fcode SUPER  
\_PROG 0f000000h

map map\_over modify delete display deMMUer end

Figure 4-20. Sample Overlay Mapping #2

The entries in the memory map correspond to the following address spaces:

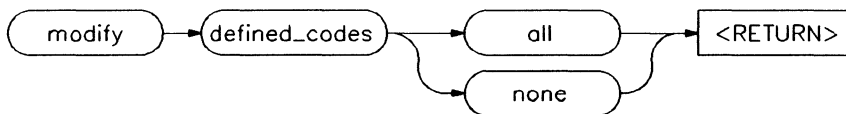
1. User application data space
2. User application program space
3. Exception vector table space
4. Emulation monitor data space
5. Exception vector table space
6. Emulation monitor program space
7. CPU\_SPACE

The emulation monitor data space (entry 4) has been overlaid onto the emulation monitor program space. This enables the 68030 processor to access data locations in the emulation monitor. The overlay is indicated in the OVERLAY column of the memory map display for entry 4. The "(6)" indicates that entry 4 is overlaid onto entry 6. The address f000000H is the address in entry 6 that corresponds to the starting address of entry 4. This memory map shows you a typical 68030 memory map.

### Using The Modify Command

The modify command lets you modify the memory map. The modify defined\_codes command lets you selectively enable or disable the 68030 function code signals (FC0 through FC2). The modify <ENTRY> command lets you modify the range, attributes, fcode, and overlay parameters of a memory map entry. The modify default command lets you change the default memory parameters.

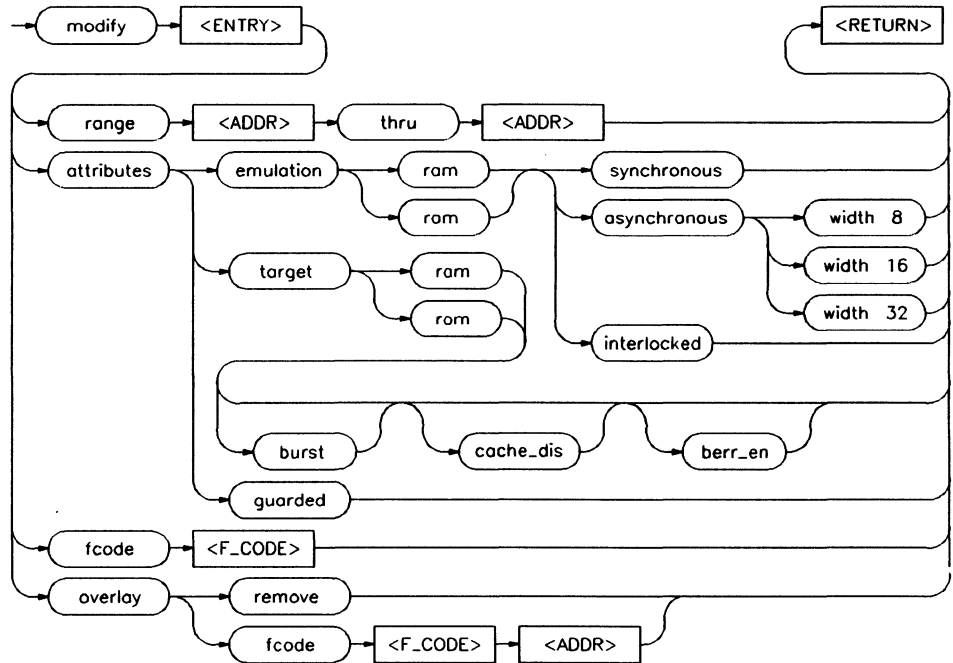
**Modify Defined\_Codes.** The modify defined\_codes command lets you selectively enable or disable the 68030 function code signals. The command syntax is shown in the following diagram:



where:

- all** enables the memory mapper to use all three function code lines (FC0 through FC2) in mapping memory. If all is selected, you can specify any of the eight function code states except CPU\_SPACE. The function codes SUPER\_PROG, SUPER\_DATA, USER\_PROG, AND USER\_DATA can be entered from softkeys. The remaining function codes must be entered as numeric values. Function code 3 is user definable. Function codes 0 and 4 are reserved for use by the processor manufacturer. Function code 7 specifies CPU address space. If you enter fcode 3, **USER\_RSVD** is displayed in the FUNCTION CODES column of the memory display. If you enter fcode 0 or 4, **MOT\_RSVD** is displayed in the FUNCTION CODES column.
- none** disables all three function code lines. when none is selected, the emulator memory mapper ignores the function code lines and monitors only the 32-bit address bus during emulation. With none selected, the fcode parameters are not available in the emulation commands. The FUNCTION CODES column is deleted from the memory map display.

**Modify <ENTRY>.** The modify <ENTRY> command lets you modify the range, attributes, fcode, and overlay parameters of an existing memory map entry. The command syntax is shown in the following diagram:



where:

**range** lets you specify a new range for the memory map entry (<ADDR> thru <ADDR>).

**attributes** lets you change the entry to:  
 emulation memory, RAM or ROM, interlocked,  
 synchronous, asynchronous with a data port width  
 of 8-bits, 16-bits, or 32 bits

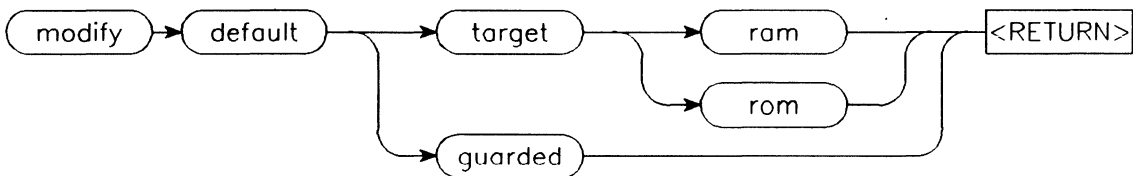
target memory, RAM or ROM, bus error blocked,  
 cache disabled, burst mode blocked

guarded

**fcode** lets you modify the function code address mapping for the entry. The selections available to you depend on the definition of the `defined_codes` parameter.

**overlay** lets you remove an overlay from an entry, e.g., the entry is converted to the physical address corresponding to the address specified in the entry, or it lets you change the function code or address range of the address space being overlaid.

**Modify Default.** Any address ranges which are not mapped when the mapping session is terminated are assigned the memory attribute specified as the default. The default attribute can be set up to be target RAM, target ROM, or guarded by using the `modify default` command. Initially, the system assigns all unmapped memory to guarded memory. The command syntax is shown in the following diagram:



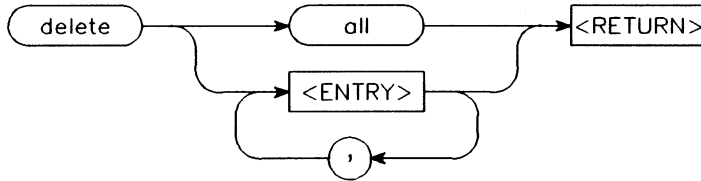
where:

**target** designates memory supplied by your target system. When `default` is mapped to `target`, the attributes are set to: caches enabled, burst enabled, and BERR enabled.

**guarded** designates an address range which is not expected to be accessed. Any processor access to a location within such a range will result in a break of the program execution.

## Deleting Memory Map Entries

Any one or all of the memory map entries can be removed by using the delete command with the exception of the default CPU\_SPACE entry. The syntax for the delete command is shown in the following diagram:



## Modify the DeMMUer Configuration

You can modify the DeMMUer configuration while in the "modify memory configuration" environment. To modify the deMMUer configuration you must press the **deMMUer** softkey. The **configure\_deMMUer** label will appear on the command line. Press the **Return** key. The display will be as shown in figure 4-21. The deMMUer is described in some detail in chapter 5 and fully in the *68030 Internal Analysis User's Guide*.

```
deMMUer configuration

deMMUer hardware disabled

Translation Control - 00000000H
  <e  sre fcl ps   is tia tib tic tid>
    0   0   0  ----  0   -   -   -   -

Virtual Address start - 00000000H

Root Descriptor Type -Invalid Descriptor

Range List -      Start          End
A             undefined range
B             undefined range
C             undefined range
D             undefined range

STATUS:  Configuring DeMMUer _____ ...R....
configure_deMMUer

range  enable  disable  set          display  return
```

**Figure 4-21. Modify DeMMUer Configuration Display**

**Ending The Mapping Session**

The memory map configuration session is exited by pressing the **endsoftkey** followed by **Return**.



## Modifying The Emulation Pod Configuration

The following question asks you whether or not you want to modify the current emulation pod configuration.

### Note



---

If you answer **no** to the "Modify emulator pod configuration?" question, the sequence will skip to the "Modify simulated I/O configuration?" question.

---

### Modify emulator pod configuration? no (yes)

no The emulation pod configuration questions are skipped and the emulation module uses the current pod configuration. The emulator will skip to the "Modify simulated I/O configuration?" question. The default pod configuration is as follows:

<b>In-circuit emulation session?</b>	<b>no</b>
<b>Disable on-chip cache</b>	<b>yes</b>
<b>MMU enabled during session</b>	<b>no</b>

yes You must answer the following emulator pod configuration questions in order to reconfigure the emulator pod.

## Configuring for In-circuit Emulation Session

Figure 4-22 shows the display that appears with the following question.

### Note



---

If you answer **no** to the "In-circuit emulation session?" question, the sequence will skip to the "Disable on-chip cache?" question.

---

In-circuit emulation session

-----  
The emulator may be adjusted to the target system by controlling  
DMA and the processor clock rate.

Out of circuit emulation session

-----  
The clock rate is locked at 20 MHz.

STATUS: Configuring M68030 emulator pod \_\_\_\_\_ .....

In-circuit emulation session? no

yes

no

RECALL

**Figure 4-22. Configuring for In-circuit Emul. Display**

**In-circuit emulation session? no (yes)**

- no The emulator is configured out-of-circuit. As such the clock rate is set to 20MHz. This question has no other action than to control whether clock or DMA questions are asked next. The question does not force the emulator to be used either in or out of circuit.
- yes The emulator is configured in-circuit, operating with target hardware. As such the emulator may be adjusted to the target system by controlling the DMA and clock rate.

## Enabling DMA Transfers

Figure 4-23 shows the display that appears with the following question.

### Note



---

If you answer **no** to the "Enable DMA transfers?" question, the sequence will skip to the "CPU clock rate faster than 25 MHz?" question.

---

Enable DMA transfers? no (yes)

```
DMA transfers enabled
-----

Bus requests will be admitted to the processor.  If the AS, address,
and data lines are active at the processor pins during the DMA cycles,
the analyzer will capture those states.

DMA transfers disabled
-----

Bus requests are blocked to the processor.

STATUS:  Configuring M68030 emulator pod _____.....
Enable DMA transfers? no

yes      no                                     RECALL
```

Figure 4-23. Enable DMA Transfers Display

- no Bus requests are blocked to the processor. The processor ignores the BR and BGACK input signals and does not respond with BG.
- yes Bus requests are admitted to the processor. If the AS, address, and data lines are active at the processor pins during DMA cycles, the analyzer will capture those states. The processor responds normally to the assertion of the BR (Bus Request) and BGACK (Bus Grant ACKnowledge) signals.

## Enabling DMA Transfers Into Emulation Memory

### Note



---

If you answered **no** to the previous question, this question is not displayed on your screen.

---

### Enable DMA transfers into emulation memory? no (yes)

- no DMA transfers to memory addresses mapped as emulation memory are disabled.
- yes DMA transfers to memory addresses mapped as emulation memory are enabled. The DMA device must generate all required control signals (AS, DS, R/W, SIZ, etc.) and meet the 68030 timing specifications.

## CPU Clock Rate Determination of Wait States

Figure 4-24 shows the display that appears with the following question.

### CPU clock rate faster than 25 MHz? no (yes)

CPU clock rate

-----  
If the external clock rate is greater than 25 MHz, six wait states will be added to emulation memory accesses. If the external clock speed is less than or equal to 25 MHz, four wait states will be added for emulation memory accesses.

STATUS: Configuring M68030 emulator pod \_\_\_\_\_ .....  
CPU clock rate faster than 25MHz? no

yes

no

RECALL

**Figure 4-24. CPU Clock Rate Display**

- no If the clock rate is less than or equal to 25.0MHz, all emulation memory accesses will occur with four wait states.
- yes If the clock rate is greater than 25.0MHz, six wait states will be inserted for emulation memory and interlocked emulation memory accesses.

## Disabling On-chip Cache

Figure 4-25 shows the display that appears with the following question.

### Disable on-chip cache? yes (no)

- no      If the cache is enabled by answering no to this question and is also enabled by the target system hardware, the analyzer may not show all memory accesses (the analyzer cannot detect cache hits). You must answer yes to this question in order to use all of the analysis features.

```
On-chip cache disabled
-----

Analysis functions perform normally.

On-chip cache enabled
-----

The cache may be enabled by the target.  If this is done, then:

* the analyzer will not be able to capture a cache hit, as
  this access is not apparent on the processor pins
* entry into the foreground monitor causes monitor instructions
  to be stored in the cache

STATUS:  Configuring M68030 emulator pod _____ .....
Disable on-chip cache? yes

yes      no                                     RECALL
```

Figure 4-25. Disable On-chip Cache Display

yes      The processor caches are enabled. A yes answer improves system performance but much analysis capability is lost.

The enable (E) bits of the CPU CACR register must be set by the target software for the caches to be enabled.

The 68030 has both program and data cache with separate enables in the CACR. Refer to chapter 6, "Target System Interface", for more information regarding the on-chip cache.

### Enabling MMU For Use During Emulation Session

Figure 4-26 shows the display that appears with the following question.

```
MMU enabled during session
-----

This option does not explicitly enable the MMU, but rather allows
the target to enable the MMU. If the user intends to use the MMU
during the emulation session, this option should be chosen.

MMU disabled during session
-----

The hardware will pull on the MMUDIS line and so the MMU can not be
enabled by the target system. Certain emulation features (e.g.
memory access or analysis) can optimize behavior (or provide greater
functionality) by knowing that logical addresses equal physical
addresses. This selection will disable the deMMUer hardware.

STATUS:  Configuring M68030 emulator pod _____ .....
MMU enabled during session? yes

yes      no                                     RECALL
```

Figure 4-26. MMU Enabled During Session Display

**MMU enabled during session? no (yes)**

- no      The emulator disables the internal MMU.
- yes      If the MMU is enabled, the keywords *logical* and *physical* are meaningful for memory access. The *deMMUer* configuration (described during memory configuration) will be loaded. The target system is expected to enable the MMU hardware and set up all of the translation tables, root pointers, etc.

**Modifying Simulated I/O Configuration**

The simulated I/O subsystem must be set up by answering a series of configuration questions. These questions deal with enabling simulated I/O, setting the control addresses, and defining files used for standard I/O.

**Modify simulated I/O configuration? no (yes)**

Answering yes to this question causes a series of simulated I/O questions to be asked. For information on how to answer these questions to configure your system, refer to chapter 9 of this manual. For additional information about simulated I/O, refer to the *Simulated I/O Reference Manual*.

Answering no to this question bypasses all other simulated I/O questions.

**Modifying Simulated Interrupt Configuration**

Simulated interrupts are enabled by answering a series of configuration questions.

**Modify simulated interrupt configuration? no (yes)**

If you answer yes, the simulated interrupts questions will be asked. If you answer no, the questions will be skipped. Simulated interrupts enable you to write and test software which depends upon the occurrence of preemptive interrupts using an emulator that is out of circuit. Information describing how to configure your system for simulated interrupts is contained in chapter 9 of this manual.



## Naming The Configuration File

This question lets you name an emulation configuration file containing the emulation configuration information you have just entered. The configuration file is stored on disc and can be called up for use during a future emulation session.

### Configuration file name?

Type in the filename you want and press **Return**.

If you press **Return** without entering a name, the current emulation session will be configured as you specified in your answers and the information will be saved as the new default configuration of the emulator. To restore the original default file provided with the emulation software, you must reinitialize the HP 64120A Cardcage.

### Note



---

If you assign a new name to the configuration file and you are using a command file to enter your emulation session, remember to modify your command file to change the name of your emulation configuration file (refer to the *HP 64000-UX User's Guide* for more information relating to command files).

---

### Note



---

Emulator configuration files are slot dependent. Use of a given configuration file on one emulator and subsequent reuse on an emulator in another cardcage slot will result in the message "Bad Module File". This message indicates that the configuration file specified was not associated with the current emulator. The message is displayed as a warning only. The emulator software will automatically rebuild the configuration file with correct cardcage slot information for the current emulator.

---

---

## Notes



## DeMMUer - What It Is And How It Works

---

### Overview

This chapter:

- Describes what the deMMUer is.
- Describes how the deMMUer operates
- Describes when to use the deMMUer.
- Describes when not to use the deMMUer.
- Describes the conditions under which the deMMUer will not perform reverse-address translations.
- Describes the restrictions associated with deMMUer operation.
- Describes when to start the deMMUer.
- Describes how to turn the deMMUer on and off.
- Describes the deMMUer configuration setup.
- Describes how to access the deMMUer configuration display.

---

## Introduction

You will need to read this chapter only if you are using the MMU of the 68030 and your internal analyzer is operating from input supplied by a deMMUer board. If you are not using the 68030 MMU active mode, no address translations occur, and you can ignore this chapter.

### Note



---

For more specific information on the deMMUer, refer to the *68030 Internal Analyzer User's Guide*, especially for detailed instructions on how to set up the deMMUer.

---

---

## What The DeMMUer Is

The DeMMUer consists of hardware and software that is used to facilitate the display of trace data when the 68030 MMU is active. Without the DeMMUer, the analyzer only has access to the physical bus, so only physical addresses (that is; no symbols, source reference, etc.) would be displayed. The deMMUer tracks MMU table walks to get the latest logical-to-physical translation information. As a result, the deMMUer is able to effectively translate the physical address information to logical. This allows the analysis display software to perform symbol lookups on the addresses (as well as source referencing).

---

## How The DeMMUer Operates

The on-chip Memory Management Unit (MMU) of the 68030 translates logical (virtual) addresses to physical addresses that are placed on the processor address bus. The deMMUer translates the physical addresses back to logical addresses in real-time. The deMMUer tracks only the physical addresses in the ranges specified in the deMMUer configuration display.

The physical address from the 68030 MMU is supplied as an input to the deMMUer. The deMMUer contains a set of translation tables like those in the 68030 MMU. The deMMUer translation tables provide the reverse function of the translation tables in the MMU (given a physical address, they look up the logical address from which it was derived). The deMMUer outputs the logical address corresponding to the physical address from the 68030 MMU.

Each time the 68030 MMU performs a table search, the deMMUer detects the event and follows MMU activity to build a corresponding set of tables for its reverse-address translations.

If you have the deMMUer running from the time you start the 68030 MMU, the deMMUer will have current translations to reverse each of the translations performed by the MMU.

**Note**



---

Be sure to flush the address translation cache (ATC) of the MMU before enabling the MMU. Otherwise, out-of-date translations (logical to physical) may reside in the ATC. There is no facility in the 68030 emulator/analyzer to flush the ATC. You can include an option to the command that loads the TC register or loads the root pointer to ensure that the ATC is flushed after reset.

---

For addresses for which the deMMUer has no translation, it supplies the physical address that was output by the 68030 MMU, and tags it as being a physical address. The analyzer will show this address in its trace list, but it will not be able to show any symbol associated with this address, nor will it be able to recognize any trace commands occurring on this address if those commands are specified using source-file symbols.

---

## When To Use The DeMMUer

You need to use the deMMUer when the 68030 MMU is active, and you want to use any of the following features during analysis of a program:

1. You want the trace list to show the assembly language form of the activity it captured during a trace. The inverse assembler requires sequential logical addresses in order to look up the next piece of program information. Physical addresses will probably be non-sequential when crossing a page boundary.
2. You want to enter a trace specification that will be met when a certain source-file event appears during a trace. To do this, you enter the name of the source-file symbol that identifies that event. Basic trigger/store/count features are not supported for code in physical addresses. The symbols in a file are always logical. In a dynamic environment, the relationship between an instruction or data location and its physical address may not be constant throughout the running of a program.
3. You want the trace list to show address values in terms of the symbol names assigned in the source files. Symbol and source line referencing operates on the fact that a symbol or source line resides at a particular logical address. That relationship is established with the language tools. The source referencing has no knowledge of physical addresses.
4. You want to perform high-level analysis on the program you are developing by using such tools as the HP Software Performance Measurement Tool (SPMT). High-level analysis tools, such as SPMT, gather data based on logical address information. These tools have no facilities for performing physical-to-logical address translations.

---

## When To Turn Off The DeMMUer

Turn off the deMMUer when you want to trace activity that shows the addresses within physical memory. This information may be useful when you are analyzing the behavior of your operating system.

---

## Under What Conditions The DeMMUer Will Not Perform Reverse-Address Translations

There are two conditions under which the deMMUer will not perform reverse-address translations:

1. If the root pointers use **page** descriptor DT fields. In this case, no table searches will occur. Physical addresses will equal logical addresses plus the offset specified in the root pointer.
2. If the two root pointer Descriptor Type (DT) fields are different types (for example, one short and the other long), and both root pointers are used, the deMMUer will fail to work because the deMMUer has facilities for only one root-pointer definition. Refer to how to select a root pointer descriptor type in the *68030 Internal Analyzer User's Guide* for suggestions of how to handle this problem.

---

## When To Start The DeMMUer

You can start the deMMUer at the same time as the 68030 MMU starts, or you can turn on the deMMUer after the MMU has been operating. Each case is discussed in the following paragraphs:

### Startup With The Emulator

The best time to start the deMMUer is just before beginning a run of program. The deMMUer flushes its reverse translations as part of the processor reset procedure. This ensures that the translation tables within the deMMUer contain no old translations. Then the deMMUer waits to detect the first table search performed by the 68030 processor. Logical address information is available immediately after reset. All table searches are monitored, keeping the deMMUer physical-to-logical address translations up to date.

### The Emulator Was Running Without Using The DeMMUer, And Now I Want To Use It

If the deMMUer was configured and enabled properly prior to running your program, the deMMUer may be turned on (by configuration or by the **set analysis mode logical** command) later, and will contain the current reverse translations.



---

## How To Turn On And Turn Off The DeMMUer

There are two ways to turn on and turn off the deMMUer: one is by setting the analysis mode, and the other is by invoking the emulation configuration set of questions. Each is described below.

### Note



---

You may turn on the deMMUer and still have only physical address information. The deMMUer can only supply logical address information after you have (1) enabled the MMU of the 68030 processor, (2) set up a valid deMMUer configuration, and (3) enabled the deMMUer. The way to set up the deMMUer configuration display and enable the deMMUer is discussed in the *68030 Internal Analyzer User's Guide*.

You will always have logical addresses when the 68030 MMU is off.

---

## Turn On/Off By Using Emulation Configuration Questions

Invoke the emulation configuration questions by using the **modify configuration** command. Proceed through the questions until the following configuration question can be answered, then answer it **yes**:

Modify memory configuration?

In the memory mapping display, enter the following command:

```
configure_deMMUer
```

In the deMMUer configuration display, enter the following command:

```
enable_deMMUer
```

Even though you have activated the deMMUer, it will still provide physical address information for analysis until it has been loaded with a valid configuration.

Once turned on, the deMMUer will track the MMU activity, and update its translation tables each time the MMU makes a change to its translation tables. Note that the MMU is turned on or off by

another emulation configuration question that appears after the memory mapping display:

"MMU enabled during session? **yes**"

**disable\_deMMUer**

This turns off the deMMUer. Only physical addresses will be supplied to the analyzer. Therefore, only the physical analysis mode will be available.

**Turn On/Off By  
Setting The Analysis  
Mode**

If you have the 68030 MMU enabled, and you have a valid value in the Translation Control (TC) register of the deMMUer configuration, and you have enabled the deMMUer, then you can turn on the deMMUer from within an emulation session, by entering the following commands:

**set analysis mode logical**

This turns on the deMMUer, providing logical addresses to the analyzer. The analyzer uses these addresses to perform symbol searches to satisfy trace specifications and show symbols in trace lists.

**set analysis mode physical**

This turns off the deMMUer. Physical addresses will be supplied to the analyzer. The trace lists will show the physical addresses, but the analyzer will not accept or display source-file symbols.

---

**DeMMUer  
Configuration  
Setup**

The deMMUer default configuration display is shown in figure 5-1. You must set up this configuration with valid entries before the deMMUer can perform its reverse address translations. Setup instructions for the 68030 deMMUer are described in the *68030 Internal Analyzer User's Guide*.

```
deMMUer configuration

deMMUer hardware disabled

Translation Control - 00000000H
  <e  sre fcl ps  is tia tib tic tid>
    0  0  0  ----  0  -  -  -  -

Virtual Address start - 00000000H

Root Descriptor Type -Invalid Descriptor

Range List -      Start          End
A              undefined range
B              undefined range
C              undefined range
D              undefined range

STATUS:  Configuring DeMMUer_____...R....
configure_deMMUer

range  enable  disable  set          display  return
```

Figure 5-1. DeMMUer Configuration Display

---

## How To Access The DeMMUer Configuration Display

Invoke the emulation configuration questions by using the **modify configuration** command. Proceed through the questions until the following configuration question can be answered:

Modify memory configuration? **yes**

In the memory mapping display, enter the following command:

**configure\_deMMUer**

In the deMMUer configuration display, you can turn the deMMUer on or off and define values and ranges to be used by the deMMUer during its operation. The procedures you follow to make these entries are discussed in the *68030 Internal Analyzer User's Guide*.

When you are finished configuring the deMMUer, return to the memory mapping display by using the **return** command. With a valid configuration setup, the deMMUer will be able to perform its reverse address translations.

# Target System Interface

---

## Overview

This chapter provides information on:

- 68030 pins and how the emulator pod interacts with those pins.

It also provides information on the appropriate use of the following emulator and processor features when the emulator is used with a target system (in-circuit emulation):

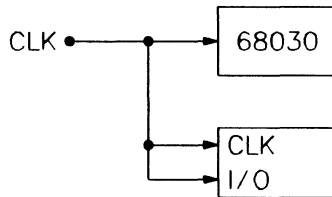
- Emulation and target system  $\overline{DSACK}$  and  $\overline{STERM}$  signals
- Vector base register
- The internal 68030 caches
- Using function codes for displaying and modifying reserved address space
- Enabling/disabling the bus error signal ( $\overline{BERR}$ )
- Using DMA
- Using the run from ... until command
- Using the emulation foreground monitor
- Memory access timing issues
- Loading absolute files.

Read this chapter before attempting to connect and operate the emulator with your target system.

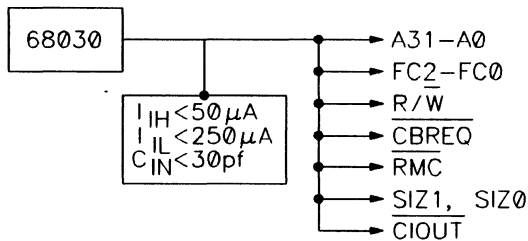
## 68030 Signals

The following section discusses each of the 68030 pins and how the pod interacts with each of those pins. For a summary of the timing specifications, and the AC and DC specifications of the 68030 emulator refer to appendix C. In this section, the emulator/target electrical interface is shown for each signal. The interface diagram is either with the signal definition or is referenced to a signal that has an identical interface. All circuitry referred to is on the active probe.

**CLK** The clock signal line is unbuffered to the 68030 processor so that synchronous timing relationships are maintained. The emulator is more of a load on the clock signal than the 68030 processor. For the timing specifications given in appendix C to be valid, the clock signal must meet the rise and fall time of specifications 4 and 5 in appendix C.



**A(31-0)** The 68030 address bus is not buffered to the target system. The emulator loads these signals so that the amount of capacitance they can drive is reduced.



**FC2-FC0** The Function Code (FC2-FC0) lines are not buffered to the target system. The emulator loads these signals so that the amount capacitance they can drive is reduced. The emulator/target electrical interface is the same as shown for the address bus.

**$\overline{R/W}$**  The Read/Write line is not buffered to the target system. The emulator loads this signal so that the amount capacitance it can drive is reduced. The emulator/target electrical interface is the same as shown for the address bus.

**$\overline{CBREQ}$**  The Cache Burst Request line is not buffered to the target system. The emulator loads this signal so that the amount capacitance it can drive is reduced. The emulator/target electrical interface is the same as shown for the address bus.

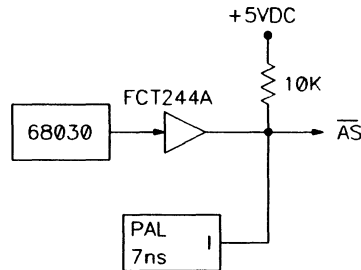
**$\overline{RMC}$**  The Read-Modify-Write Cycle line is not buffered to the target system. The emulator loads this signal so that the amount capacitance it can drive is reduced. The emulator/target electrical interface is the same as shown for the address bus.

**SIZ0-SIZ1** The Size signal lines are not buffered to the target system. The emulator loads these signals so that the amount capacitance they can drive is reduced. The emulator/target electrical interface is the same as shown for the address bus.

**$\overline{CIOUT}$**  The Cache Inhibit Out signal line is not buffered to the target system. The emulator loads this signal so that the amount capacitance it can drive is reduced. The emulator/target electrical interface is the same as shown for the address bus.

## AS

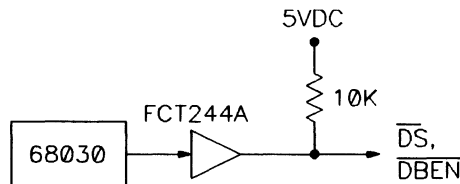
The 68030 Address Strobe signal line is buffered by the emulator prior to going to the target interface. AS is driven to the target when the processor is running except when the emulator is in the background monitor, or when the bus has been relinquished. The AS signal from the target system is also treated as an input during DMA so that emulation memory and the analyzer can see those cycles.



Buffering the AS signal causes it to be delayed to the target system. This delay may be significant in some systems, but in a system which has AS heavily loaded, it may not even be noticeable.

## DS, DBEN

The 68030 Data Strobe and Data Buffer Enable signal lines are buffered by the emulator prior to going to the target interface. DS and DBEN are driven to the target when the processor is running except when the emulator is in background monitor, or when the bus has been relinquished. Buffering these signals causes them to be delayed to the target system.



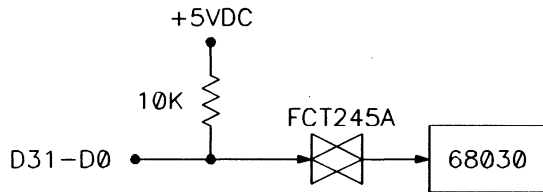


## ECS, OCS

The 68030 External Cycle Start and Operand Cycle Start signal lines are buffered by the emulator prior to going to the target interface. ECS and OCS are driven to the target when the processor is running except when the emulator is in background monitor (when they are optionally driven), or when the bus has been relinquished. Buffering these signals causes them to be delayed to the target system. The emulator/target electrical interface for these signals is the same as shown for the DS signal.

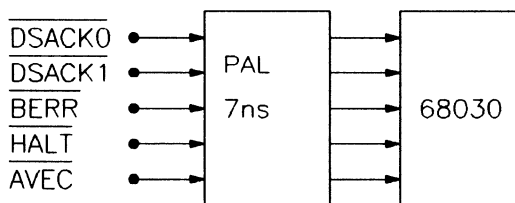
## **D(31-0)**

The data bus is buffered between the 68030 and the target interface. The buffers only drive the target system during write cycles mapped to target memory, or during read cycles in DMA that are mapped to emulation memory. The processor receives the data from the target system when a read cycle is mapped to target memory during normal program operation. Buffering the data bus lines causes the emulator to require more setup time than the processor.



## DSACK1-DSACK0

The Data Transfer and Size Acknowledge signals are buffered between the target system and the 68030. The signal from the target system is only sent to the processor during normal cycles mapped to target memory, during interlocked emulation memory cycles, or during foreground data space cycles. During all interlocked or monitor cycles and during emulation jams, the DSACK once asserted, is forced to a 32-bit access. During interlocked emulation memory cycles, the target DSACK signals are not allowed until the emulation memory has data valid. Buffering the DSACK lines causes the emulator to require more setup time than the processor.



## BERR

The Bus Error signal is buffered between the target system and the 68030. The BERR signal can be blocked from going to the processor during target cycles and/or emulation memory cycles. It is always blocked during monitor and other special emulation cycles.

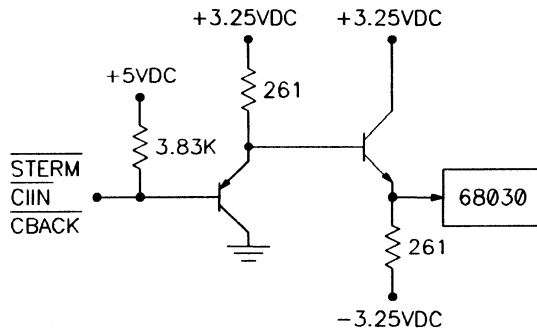
If the HALT signal is asserted at the same time as the BERR signal, the BERR signal is not treated as a bus error; but as a retry. Retry cycles are never blocked by the emulator. The emulator/target electrical interface is the same as shown for the DSACK signals.

## HALT, AVEC

The Halt and Autovector signals are not blocked by the emulator. The emulator/target electrical interface for these signals is the same as shown for the DSACK signals.

## **STERM**

The Synchronous Termination signal is buffered between the target system and the 68030. The signal from the target system is only sent to the processor during normal cycles mapped to target memory, during interlocked emulation memory cycles, or during interlocked foreground data space cycles. During interlocked emulation memory cycles, the target STERM signal is not allowed until the emulation memory has data valid. Buffering the STERM signal line causes the emulator to require more setup time than the processor.



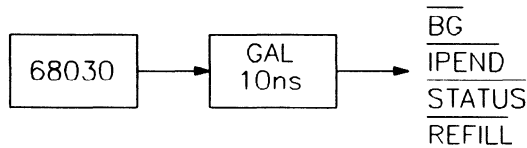
## **CIIN**

The Cache Inhibit In signal is buffered between the target system and the processor. It is not blocked by the emulator. The emulator/target electrical interface for these signals is the same as shown for the STERM signal.

## **CBACK**

The Cache Burst Acknowledge signal is buffered between the target system and the processor. The signal is blocked except during normal target cycles mapped to allow bursting. The emulator/target electrical interface is the same as shown for the STERM signal.

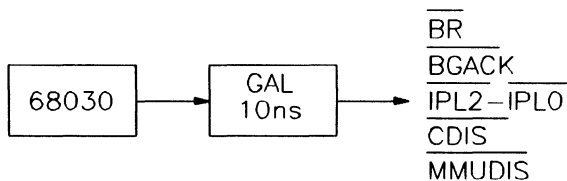
**BG** The Bus Grant signal is buffered between the processor and the target system. The signal is blocked when DMA is disabled.



**IPEND** The Interrupt Pending signal is buffered between the processor and the target system. The signal is blocked during interrupts caused by the emulator break facility. Whether or not the signal is blocked can be configured. Refer to chapter 4 for configuration information.

**STATUS, REFILL** The Microsequencer Status and Pipe Refill signals are not blocked by the emulator. The emulator/target electrical interface for these signals is the same as shown for the BG signal.

**BR, BGACK** The Bus Request and Bus Grant Acknowledge signals are buffered between the target system and the processor. These signals are blocked when DMA is disabled.



**IPL2-IPL0** The Interrupt Priority Level signals are buffered between the target system and the processor. These signals are blocked while the emulator is in the background monitor. Use of the emulator foreground monitor can delay handling of an interrupt. This is because the emulation monitor is an interrupt handler itself and maskable interrupts are normally disabled during execution of the



---

## Emulation And Target System DSACK and STERM Signals

### Interlocking Emulation Memory DSACK and STERM and Target DSACK and STERM Signals

If your target system memory requires wait states, you should interlock the emulation memory DSACK and STERM signals with the target system DSACK and STERM signals. This causes accesses to emulation memory and accesses to target memory to properly reflect system performance when the emulator is removed.

#### Note



---

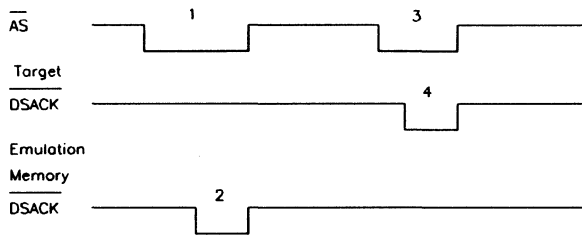
When operating the emulator at 25 MHz, four wait states will be added **EVEN** if the target system responded with a zero-wait-state termination during interlock operation. At 33 MHz, the emulator adds six wait states to emulation memory accesses.

---

If target system memory requires wait states, the first target memory access after an emulation memory access may fail if emulation and target DSACK and STERM signals are not interlocked. See the timing diagram in figure 6-1.

The following rules should be used to determine whether or not to interlock emulation and target DSACK and STERM signals.

1. If the target system generates DSACK and STERM signals for all emulation memory address ranges, emulation and target DSACK and STERM signals should be interlocked.
2. If the target system does not generate DSACK and STERM signals for a range of emulation memory, emulation and target DSACK and STERM signals must not be interlocked.



1. An access to emulation memory.
2. Emulation memory DSACKs terminate cycle properly.
3. Access to target memory.
4. Target DSACKs from emulation memory accesses (1) prematurely terminate the cycle before correct data is available from target memory.

**Figure 6-1. Memory Access Timing, No DSACK Interlock**

3. If there is no target system (that is; out-of-circuit emulation), DSACK and STERM signals cannot be interlocked.

Each block of emulation memory can be individually interlocked during emulation configuration.

### **DSACK and STERM Signal Problems In Target Systems**

Many target systems violate 68030 DSACK and STERM signal specifications. These violations are usually marginally acceptable to the 68030 CPU in the target system, but cause problems when the emulator is plugged in. These specification violations usually result in improper data fetches from memory and cause target system failure with the emulator installed.

### **Use Of Open Collector Drivers**

One of the most common problems is associated with the use of open-collector drivers on the DSACK and STERM lines. DSACK and STERM lines often have pullup resistors that pull the DSACK and STERM signals high at the termination of a memory cycle.

Improper values for pullup resistors can cause DSACK and STERM signals not to be pulled up fast enough and may interfere with the next cycle. This occurs when the pullup resistor value is too large to return DSACK and STERM to a proper high level before the next cycle begins. In this case, the still low DSACK and STERM signal causes a premature termination of the second cycle, resulting in improper data fetches by the CPU.

### **Early Removal Of DSACK Signals**

Some target system designs do not adhere to the 68030 specification which states that the DSACK signals must not be removed prior to the negation (low to high transition) of the address strobe at the end of a cycle. In the simplest case, this results in "no DSACK" messages appearing in the tracelist, which in turn causes inverse assembly failure. More seriously, the emulator may completely malfunction depending on how early the DSACK signal is removed prior to address strobe transition.

### **Isolating The DSACK Problem**

If you suspect that your target system may have either of the preceding problems, use a timing analyzer to help isolate the problem. Take a trace of the CPU clock, address strobe, data strobe, and the DSACK signals during the failing cycle (use the BNC's on the back of the HP 64120 cardcage to drive the trigger, if possible). Examine the results and compare your findings to the electrical specifications of the 68030 processor and the HP 64430 emulator.



---

## Using The Vector Base Register

The 68030 CPU gets exception vectors from the exception vector table located at the address contained in the **Vector Base Register (VBR)**.

The 68030 emulator uses a jamming technique for breaks and software breakpoints. Therefore, the value of the VBR is not needed to perform most monitor functions. This implies that the vector table may be located anywhere without adversely affecting emulator operation.

The single-step feature, when using the foreground monitor, does require the use of the trace exception vector ( $VBR + 24H$ ). If the single-step feature is to be used, you must make sure that the trace exception vector always points to the monitor (`MONITOR_ENTRY`).

The monitor can handle various exceptions by displaying a status message, entering a loop within the monitor, and then waiting for user intervention. These exceptions include Bus Error, Address Error, Divide by zero, etc. If you use these exceptions, you must maintain the exception vector table so that the vectors in use always point to the appropriate monitor location.

---

## Using The Internal 68030 Caches

Using the internal 68030 caches affects several functions of the 68030 emulator. The following sections discuss use of the internal caches and their effect on emulator operation.

### Cache Control

When the emulator is operating out-of-circuit, the "Enable Caches?" configuration question has a different interpretation than when plugged into a target system. When using the emulator out-of-circuit, a "yes" answer to the "Enable Caches?" configuration question forces the  $\overline{\text{CDIS}}$  signal high within the pod. When using the emulator in-circuit, a "yes" answer connects the target system  $\overline{\text{CDIS}}$  signal to the emulator CPU's  $\overline{\text{CDIS}}$  input, allowing the emulator to track target system  $\overline{\text{CDIS}}$ . In both cases, a "no" answer forces  $\overline{\text{CDIS}}$  low within the emulator.

Recall also that the target system  $\overline{\text{CDIS}}$  must be high, and bit zero of the Cache Control Register (CACR) must be set to 1 for the instruction cache to be enabled and bit 8 of the CACR must be set to 1 for the data cache to be enabled.

If the target system uses the internal 68030 caches, the caches must be enabled by answering "yes" to the "Enable Internal Caches?" configuration question.

When the  $\overline{\text{CDIS}}$  signal from the target system is set to 1, the caches still are not enabled until bits 0 and 8 of the (CACR) are set to 1, as shown in the following example:

```
MOVEQ.L  S#11,D0
MOVEC    D0,CACR      ;software enable cache
```

Enabling the caches affects analysis trigger, store, count, and Global Context functions. Additionally, some program read states may be missing from the trace list.

The caches are not frozen on entry to the foreground monitor. This results in overwriting the cache contents.

If a breakpoint is set for an address currently contained in cache, the breakpoint will not be recognized until the CPU fetches from that address in main memory again. The run until command is similarly affected since breakpoints are used in the command implementation.

## Analysis with Cache

The 32-bit internal analyzer can capture any cycle that occurs external to the 68030 CPU. When caches are enabled, read cycles may occur only internal to the CPU. This is the general case with tight program loops and with high performance code segments that are frequently locked in cache. Since the analyzer cannot capture internal cycles, it has no way to display these cycles in the tracelist. This can result in missing trace data and high-level source lines, and even improper disassembly. The analyzer will also miss the occurrence of trigger, store, count, sequence or context patterns if they occur only as internal cycles.

In general, any program segment that executes from cache will generate some external cycles, the major exception being timing loops. In these cases, you may be able to select trigger and store patterns that correspond to external cycles. If no external cycles are normally generated, you may be able to place "markers" in the cached code such that the code will generate an external cycle for analysis purposes when executed.

The mapping function allows pages of target memory to have caching disabled for analysis purposes

Since the analyzer contains a high precision cycle-to-cycle timer, you can usually examine the tracelist to determine where cache execution occurred.

## Using Breakpoints With Caches Enabled

You may see situations where breakpoints do not appear to be functioning properly when the instruction cache is enabled. This can happen when you are using the "run until" command as well as breakpoint commands.

Consider the following segment of code (a simple software timing loop), and assume that the cache is enabled:

Address	Code
1000:	RELOOP NOP
1002:	NOP
1004:	NOP
1006:	NOP
1008:	NOP
100A:	SUBQ.L #1,D0 ; decrement loop counter
100C:	BNE RELOOP ; reloop if not 0

Because the instruction cache is enabled, no external memory cycles are generated for addresses 1000H thru 100CH after their initial load into cache. Breakpoints set at any cache resident address may never be encountered. This situation occurs when the CPU does not generate an external program read cycle to memory and therefore never "sees" the breakpoint that was set.

### **Target Memory Breakpoints**

Breakpoints set in target system memory differ from those set in emulation memory. If the breakpoint address is mapped to target system memory, the monitor must intervene in order to set the breakpoint. Execution of the monitor overwrites cache locations previously occupied by the user program. When the emulation monitor is exited, the user program is fetched again from memory, breakpoint included. This results in normal breakpoint behavior.

### **Emulation Memory Breakpoints**

This problem is worse when the breakpoint address is mapped to emulation memory. Due to the dual-port nature of the memory system, the host sets breakpoints in emulation memory without requiring execution of the emulation monitor. In this case, the mechanism of setting breakpoints does not clear cache and force a refetch of the newly specified breakpoint.

For breakpoints to function properly out of emulation memory, you need to clear the cache before setting or resetting the breakpoint. Do the following steps before setting a breakpoint:

1. Break to the emulation monitor program.
2. Display CPU registers.
3. Modify CACR bit C to 1 and then to 0.
4. Set the breakpoint or enter the run until command.
5. Exit the monitor by executing a run command.

When the breakpoint is hit, you can remove it from cache by adding 68030 instructions to the emulation monitor that will set and clear the CACR C bit.

The preceding comments apply to setting software breakpoints as well as disabling software breakpoints.

---

## Using Function Codes For Displaying And Modifying Reserved Address Space

When the use of function codes is enabled during a memory mapping session, the display and modify commands use the function codes specified in the command. When function codes are disabled, function code 0 is used for all memory reference commands.

Some target systems do not use function codes to differentiate between user and supervisor space or program and data space, but do decode the "reserved" address spaces (function codes 0, 3 and 4) to generate interrupts or inhibit DSACK generators. In these cases, the emulation monitor may be customized to allow the use of a non-zero function code for the display, modify, load, and store emulator commands.

This modification requires changing two assembly statements in the monitor "COPY" routine as shown in the following listing:

```

*****
*
*   command 2 ..... access user memory
*
*****
COPY
* copy parameters from CPU space (SFC and DFC were setup by MONITOR_LOOP)

* copy byte count from parameter slot 1
  MOVES.L  PARM1,D0

* copy source address from parameter slot 2
  MOVES.L  PARM2,A0

>>> * copy source function code from parameter slot 3
>>> *   MOVES.L  PARM3,D1

>>> * force user data function code
>>>   MOVES.L  #2,D1

* copy destination address from parameter slot 4
  MOVES.L  PARM4,A1

>>> * copy destination function code from parameter slot 5
>>> *   MOVES.L  PARM5,D2

>>> * force supr data function code
>>>   MOVES.L  #5,D2

* copy access mode from parameter slot 6
  MOVES.L  PARM6,D3

```

Modifications to the emulation monitor code for non-zero function code access to target system memory include adding the two new source lines shown in lower-case and commenting out 2 lines as shown in the listing. The added and modified lines are indicated by arrows (>>>).

---

## Enabling/Disabling BERR

The 68030 emulator allows the bus error ( $\overline{\text{BERR}}$ ) signal to be received or not received during accesses to emulation memory.

If the target system generates bus errors for emulation memory address ranges, the reception of  $\overline{\text{BERR}}$  should be disabled. This would normally occur if  $\overline{\text{DSACK}}$  or  $\overline{\text{STERM}}$  signals are not generated for emulation memory accesses.

If the target system generates  $\overline{\text{DSACK}}$  or  $\overline{\text{STERM}}$  signals for emulation memory accesses, then it probably does not generate  $\overline{\text{BERR}}$  for these cycles. In this case,  $\overline{\text{BERR}}$  actually indicates a failure, and should be enabled in the emulator.

---

## Using DMA

If any devices share the 68030 bus and are able to perform DMA, then DMA should normally be enabled. This enables the CPU to receive the Bus Request ( $\overline{\text{BR}}$ ) signal, generate a Bus Grant ( $\overline{\text{BG}}$ ) response signal, and receive the Bus Grant Acknowledge ( $\overline{\text{BGACK}}$ ) response from the bus requester. The handshake sequence for DMA transfers is shown in figure 6-2.

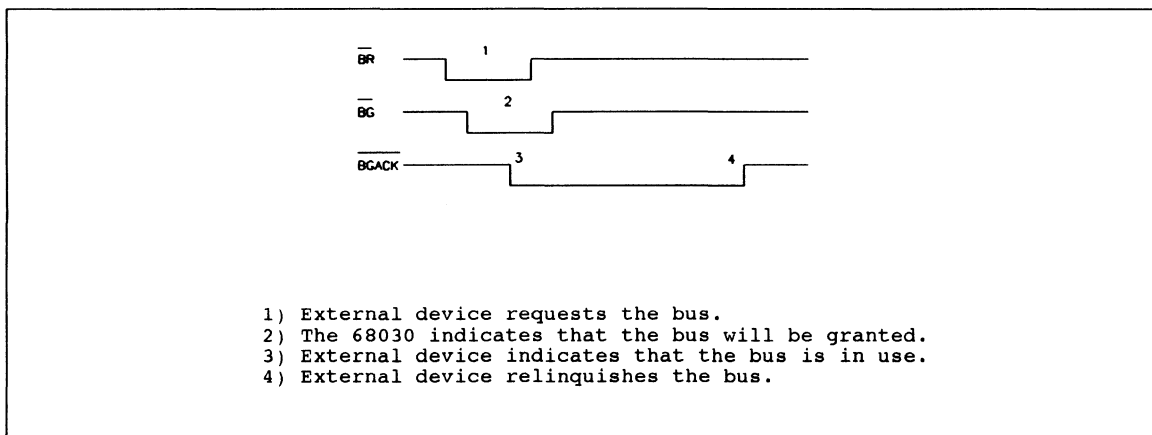


Figure 6-2. DMA Bus Request/Bus Grant Timing

If DMA is disabled, the CPU will not receive the bus request signal, and will not allow DMA cycles. This would be desirable in order to characterize system performance in a situation where DMA could not occur.

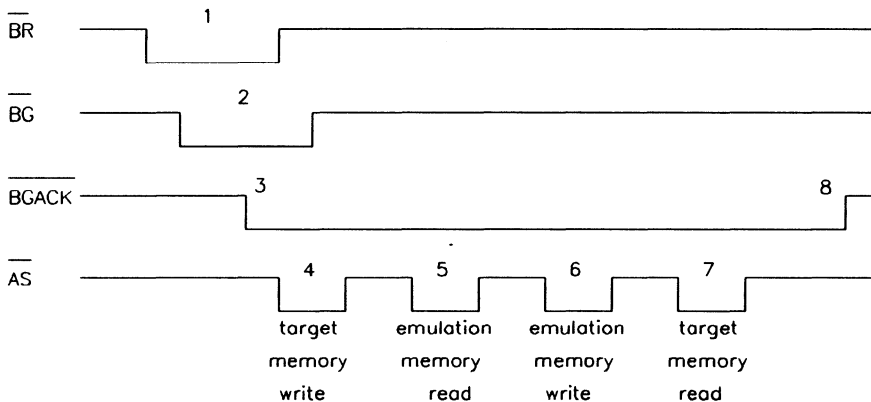
The user who has enabled DMA has a secondary option of enabling or disabling DMA to/from emulation memory.

If DMA to emulation memory is enabled, the DMA hardware has access to read from or write to emulation memory. If DSACK or STERM signals are interlocked, the DSACK or STERM signals for these accesses are supplied by the target system. The DMA master must generate cycles that conform to 68030 timing requirements.

If DSACK or STERM signals are NOT interlocked, then no DSACK or STERM signals are returned to the target system. This would cause the DMA hardware to hang if DSACK or STERM signals are required for cycle termination.

If the option to DMA to/from emulation memory has been DISABLED, the DMA cycle will still be allowed to occur, but no information will be written to, or read from emulation memory. See the timing diagram in figure 6-3.





1. DMA device requests the bus.
2. The 68030 indicates that bus will be relinquished.
3. DMA device indicates that the bus is in use.
4. DMA device generates a write with a target memory address. This cycle occurs normally.
5. DMA device generates a read with an emulation memory address. This cycle does not return valid data since DMA to emulation memory is disabled.
6. DMA device generates a write with an emulation memory address. This cycle does not modify emulation memory since DMA to emulation memory is disabled.
7. DMA device generates a read with a target memory address. This cycle occurs normally.
8. DMA transaction is complete.

**Figure 6-3. DMA Timing Diagram, DMA Disabled**

## Using The Run From ... Until Command

The run command must be used properly to avoid serious, stack related problems in the software being executed.

One of the main causes of target system "failure" while using the run command is the stack not being setup and/or restored properly by the software being executed. One common situation is for parameters to be placed on the stack prior to calling a procedure. (Parameter stacking code including the actual procedure call is usually referred to as the "calling sequence.") Assume for example that a procedure, PROC1 expects the stack frame shown in figure 6-4.

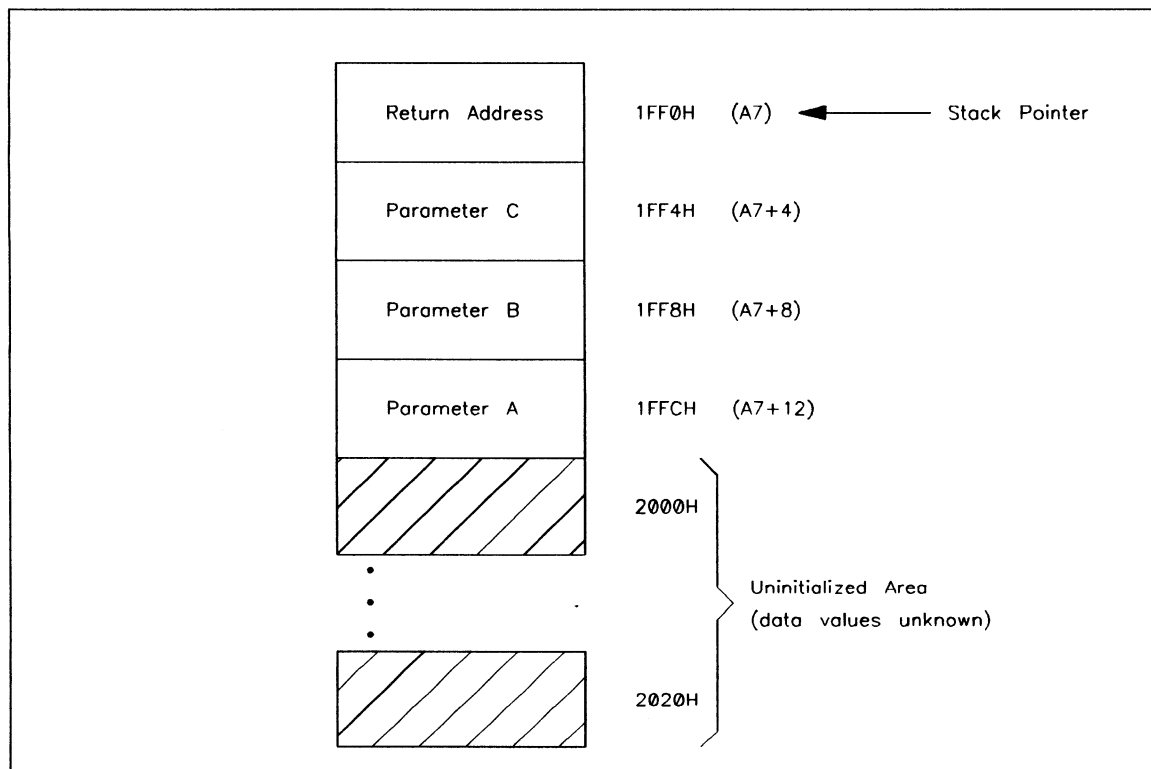


Figure 6-4. Example Stack Frame

Often, PROC1 will access the stacked parameters by referencing parameter requests to the stack pointer. This means that parameter "A" can be found at address  $A7+12$ , parameter "B" at address  $A7+8$ , etc.

If the parameters are not stacked, and/or the return address is not present, then the usual parameter references  $A7+12$ ,  $A7+8$ , etc. may reference uninitialized stack areas. Also, the return address used by PROC1 will be incorrect. This will usually result in a software failure both within PROC1 (because the parameter values are wrong) and on exit from PROC1 (because the return address was not set properly). Depending on emulator memory mapping, the "stack" areas referenced by  $A7+12$ , etc. may actually fall within guarded memory area, resulting in a guarded memory access message.

Executing the command "**run from PROC1**" prior to stacking the parameters and setting the return address is one case where this could happen. Problems also occur if a "**run from <address>**" command is executed and CPU registers, or memory locations are not properly initialized for the code to be executed at **<address>**.

Using the command "**run until**" can also cause problems. This case is different from the "run from" case in that software problems may occur on a subsequent "run" command after the "until" condition is satisfied. If a "run" command is executed after executing the "until" breakpoint, no problems should result, because the CPU will continue executing the user program from the point where it left off. If a "run from" command is executed after the "until" breakpoint, the stack, CPU registers and memory locations may be improperly set for the code to be executed at the "run from" address.

These situations cannot be corrected within the feature set of the emulator. You must be aware of your software requirements, and the mechanism used to implement the run commands. A detailed explanation of how the run command works is given in chapter 10.

---

## Using The Emulation Foreground Monitor

### Loading the Monitor

The following rules must be followed when loading the emulation monitor:

1. Both program and data spaces of the monitor must be mapped to RAM as opposed to ROM. The monitor transfer buffer, as well as many monitor "housekeeping" variables must be read and write accessible, and must therefore be mapped to RAM.

In addition, portions of the monitor must write to other monitor program locations. Since writes to ROM are always blocked, the program section, as well as the data section, of the monitor must be mapped to RAM.

2. The emulation monitor is executed in response to a level 7 interrupt. Therefore, it is always executed within supervisor space and must be located in supervisor space. If the supervisor/user function code bit is not in use, this restriction does not apply.

The emulation software recognizes only program symbols. In the case of the monitor, the symbol addresses are assumed to be associated with the SUPR\_PROG function code (since the monitor is basically an interrupt routine). Thus, when the host writes control information to, or reads information from the monitor, it must use the SUPR\_PROG function code.

### Resetting Into The Monitor

The "reset into monitor" facility of the emulator makes use of internal jamming circuitry to supply both an initial stack pointer and an initial program counter to the CPU. These values correspond to the values of monitor symbols SP\_TEMP and RESET\_ENTRY respectively. If the background monitor is being used, the initial stack pointer must be defined since stacking is done in foreground monitor.

Jamming from reset occurs only if the emulator caused the reset via the "reset" softkey. If the target system asserts the CPU reset signal, the jamming circuitry is disabled and startup from reset occurs normally, with stack pointer and program counter values being supplied from memory system addresses 0-7.

The setting of the initial stack pointer value is critical to proper system operation. Since SP\_TEMP is only provided as a small temporary stack for use with the monitor, the stack may be easily overflowed once a "run from ..." command is given, and the target system program begins execution. Portions of the monitor may be overwritten if the SP\_TEMP stack overflows.

To ensure proper operation, be sure to either extend the SP\_TEMP stack to meet target system requirements, or modify the SP\_TEMP value to point to the usual target system stack. This can be done by including an appropriate "equate" statement in the monitor, while commenting out the normal SP\_TEMP label in the monitor. For example:

```
SP_TEMP EQU <target system stack address>
```

Another approach is to be certain that software execution as a result of the "run from ..." command properly initializes the stack pointers to values appropriate to the target system.

When the emulator is in a reset condition, one of two messages appears on the emulator status line above the softkeys. If the word "Reset" appears, the present reset condition occurred as a result of the emulator. The presence of a lower case "reset" indicates that the target system is presently asserting the CPU reset signal. The 68030 emulator can be instructed to enter the emulation monitor when a "run" command is issued after "Reset" (jamming only occurs if the reset signal is asserted by the emulator). This causes the initial program counter and initial stack pointer vector to be ignored. Instead, the jamming circuitry supplies these values based on the current location of the monitor.

A possible difficulty exists if the target system performs some hardware and/or software initializations on reset. If "reset into monitor" is used, these initializations are not performed before monitor execution is begun.

---

## Memory Access Timing Issues

Access time is the time interval during a 68030 microprocessor read cycle beginning when the 68030 microprocessor places an address on the address bus and ending when valid data is present on the microprocessor's data pins.

Appendix C contains tables listing timing comparisons between the MC68030 processor and the HP 64430 emulator.

### 33 MHz 68030 Microprocessor

For a 33 MHz 68030 microprocessor running at maximum speed in synchronous mode with no wait states:

Access Time = Cycle Time + Clock Pulse Width - Specification 6 - Specification 27

Spec. 6 = Clock High to FC,Size,RMC,CIOUT,Address Valid  
= 14 ns (max),

Spec. 27 = Data-in Valid to Clock Low (Synchronous Setup)  
= 1 ns (min),

Cycle Time = 30 ns (min),

Clock Pulse Width = 14 ns (min).

Therefore:

Access Time (max) = 30 ns + 14 ns - 14 ns - 1 ns = 29 ns

### HP 64430 68030 Emulation System

For the HP 64430 68030 emulation system, the emulator adds the following delay:

Data lines buffered with a 74FCTA245 = 5 ns (max)

An easy way to calculate the maximum access time allowed by the emulator is to use the timing comparison tables provided in appendix C of this manual. The relevant worst case specifications for the emulator are as follows:

\*Access Time (max) = Cycle Time + Clock Pulse Width - Specification 6 - Specification 27

\*Specification 27 includes value added because of data line buffering shown above.

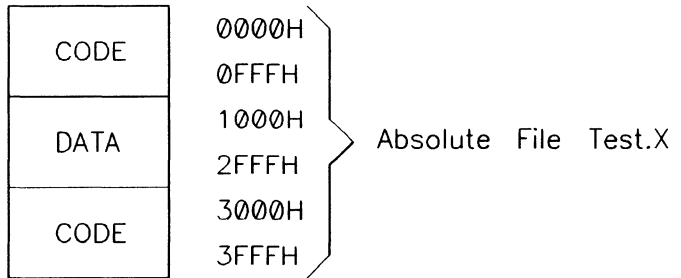
Spec. 6 = 14 ns (max)  
 Spec. 27 = 6 ns (min)  
 Cycle Time = 30 ns (min)  
 Clock Pulse Width = 14 ns (min)

Therefore:

Access Time (max) = 30 ns + 14 ns - 14 ns - 6 ns = 24 ns

## Loading An Absolute File

When an absolute file is generated, it frequently is composed of various "sections" containing code or data:



A memory map resembling that shown below might normally be generated:

Addr. Range	Attribute	Function Code
0000H - 0FFFH	EMUL RAM	SUPR_PROG
1000H - 2FFFH	EMUL RAM	SUPR_DATA
3000H - 3FFFH	EMUL RAM	USER_PROG

default = guarded

Note that upon execution of the following command, a guarded access will occur:

**load memory Test.X fcode SUPR\_PROG Return**

This is due to the fact that the "load" mechanism attempts to load the entire file using the SUPR\_PROG function code. In the case of Test.X (with the memory map above), address range 0000H - 0FFFH is mapped to emulation memory when the function code is SUPR\_PROG. The remaining address ranges of Test.X are actually mapped to GUARDED memory when the function code is SUPR\_PROG. This is because the default is set to GUARDED, and because there are no mapping definitions for SUPR\_PROG covering the remaining address ranges of Test.X.

Similar symptoms would be observed with either of the following commands:

**load memory Test.X fcode SUPR\_DATA**  
**load memory Test.X fcode USER\_PROG**

The "load memory . . ." command is defined as a vehicle to "load all memory areas" present in a given absolute file. (Guarded, as well as target and emulation memory.)

The "load memory emulation . . ." command is used to "load only areas mapped to emulation memory" in a particular absolute file.

Thus, to properly load Test.X, the following three commands would be issued:

**load memory emulation Test.X fcode SUPR\_PROG**  
**load memory emulation Test.X fcode SUPR\_DATA**  
**load memory emulation Test.X fcode USER\_PROG**

The "load memory target . . ." command is provided to "load only areas mapped to target memory" in a given absolute file.



---

## Debugging Plug-in Problems

When the emulation pod is connected to a target system, the emulator operation becomes more complex. More hardware has been added to the entire system and the user must be knowledgeable about the target system resources. The following information should be used as a guide to isolate problems that are encountered when connecting the emulation pod to a target system.

If the target system has tight timing specifications, the emulator may cause some signals to violate either the emulator 68030 or the target system timing requirements.

### Review the Configuration

An incorrect configuration file can result in improper operation. Review the entire configuration file to make sure that all of the questions are answered correctly for your target system. If you are not sure how to answer a particular question refer to chapter 4 and sections of this chapter for details concerning configuration and information about the target system interface. The command `"!more <configfilename> .EA"` can be used to view the entire configuration file.

Target systems that are able to operate without the emulation pod should be able to start with the default configuration file. This file is used whenever a new emulation session is started. The default configuration enables all of the target system signals, maps all memory as target RAM and specifies that the emulation monitor is not loaded. Verification of proper operation should be made using the internal analyzer and indications from the target system. Plug-in failures with the default configuration should be isolated before attempting to use configurations that use emulation memory or the emulation monitor. Once the default configuration works properly add emulation memory and an emulation monitor.

## **Use the Internal Analyzer**

The internal analyzer can be used with any configuration without interfering with the emulation of the processor. It passively monitors each bus cycle that the processor executes. All of the analyzer data can be displayed without disrupting the emulation process. The analyzer can be used to verify the proper operation of the program being executed and the proper operation of the hardware.

Debugging plug-in failures with the internal analyzer should start with a "trace TRIGGER\_ON a= 0h" specification before allowing the processor to run. This will capture all bus cycles starting with the reset address. Particular attention should be given to the bus size bit (B) and the data field of the first few cycles. The triggering capability of the analyzer can be used to capture conditions that are the result of a failed interface by using the "trace TRIGGER\_ON <failure\_condition>" specification. These conditions are usually incorrect code branches or status conditions such as halt or shutdown.

Failures that occur only during certain types of operations such as a CPU space address or a particular place in memory can be debugged using the capability of the analyzer to drive one of the rear panel BNC outputs or the Intermodule Bus (IMB). The trigger condition should be set up for the bus cycle in error and the trigger should be enabled to drive the BNC or IMB. These outputs can then be used with measurement tools such as timing analyzers or oscilloscopes that can be used to monitor the target system. When observing the data, keep in mind that the trigger pulse actually occurs between one and two CLK cycles after the end of the bus cycle.

## **Use the Status Messages**

Appendix C contains a complete list of emulation status line messages and their causes. Many of these conditions are not displayed unless no bus cycles have occurred for more than 250 milliseconds. If your system creates conditions that result in the 68030 not generating a bus cycle for more than 250 milliseconds, then the status message related to that condition can be ignored. Status messages such as "Write to ROM fc= <code>", "halted" and "slow device fc= <code>" provide address or status information that can be used by the analyzer as a trigger specification to trace the error condition.

## Run Performance Verification (PV)

Refer to the *HP 64430 HP-UX Hosted 68030 Emulator Service Manual* for instructions for running performance verification on the emulation system.

---

### If All Else Fails . . .

If the emulator is configured properly, and the target program and foreground monitor are loaded, unexplained behavior may still exist. This is frequently due to foreground monitor interaction with the target software and/or hardware.

In the software category, check that it is appropriate to disable interrupts while in the foreground monitor. Some systems with delta-time-interrupt structures for real-time clocks, operating system functions, etc., will crash if the delta-time-interrupt is not serviced within a preset time limit. The foreground monitor can be customized to enable or disable interrupts as required. See the "Continuing Target System Interrupts While In The Emulation Monitor" section of chapter 7.

It is possible to "disable" the normal target system function of the level 7 (NMI) interrupt through vector table modifications, and a small amount of additional foreground monitor code.

Ensure that the program being executed is not accidentally overwriting the foreground monitor or vice versa.

Use of the analyzer to examine software behavior is an effective means to solve emulation problems.

Obtain a listing of the foreground monitor and the program being executed, and use the analyzer to verify proper operation of both.

Set the analyzer to trigger on the foreground monitor entry point (MONITOR\_ENTRY), with the trigger position set to the center of the trace. This will allow you to examine CPU activity surrounding the foreground monitor entry. You can observe the stacking activity of the level 7 interrupt, as well as emulator generated jam cycles. This will enable you to determine if the foreground monitor is being initiated properly.

Ensure that the foreground monitor exits and returns to the normal program properly. Set the analyzer to trigger on the foreground monitor exit point (EXIT\_MON), and observe the unstacking as a result of the RTE instruction. Be sure that the stack contents have not been corrupted, and that the program returns to the expected location.

Remember that the use of any foreground monitor function will affect the timing of executing programs, and may be the cause of hardware and software anomalies.




# The Emulation Monitor Programs

---


## Overview

This chapter:

- Provides a comparison of the emulation foreground and background monitor programs.
- Discusses the need for, and when to use, the emulation foreground and background monitor programs.
- Discusses the break function as related to the emulation monitor programs.
- Describes the emulation foreground monitor program.
- Provides information for customizing the emulation foreground monitor.
- Describes the emulation foreground monitor memory requirements.
- Describes the emulation foreground monitor linking requirements.
- Describes the rules for loading the emulation foreground monitor.



See chapter 6, Target System Interface, and chapter 10, How the Emulator Works, for additional information about the emulation monitor and its interactions with the host computer and your target system.



---

## Introduction

The emulation monitor program is the means by which many of the emulator functions are implemented. Functions implemented with the emulation monitor are:

- Read/write target memory.
- Display/modify 68030 registers.
- Display/modify coprocessor registers.
- Execute user program.
- Break from user program by:
  - analyzer generated break
  - keyboard break
  - software breakpoint
  - jump from user program
  - memory access violation break.
- Reset into monitor.
- Single step by opcode.
- Coordinated emulation start.

A standard emulation foreground monitor source file is supplied with each emulation system. This file must be assembled and linked by the user before using. Typically, the emulation monitor is assembled and then linked with the user program to form one software module. This module is then loaded into memory. The loaded foreground monitor program enables emulation system functions to operate properly.

You can modify the emulation foreground monitor to accommodate a particular target system or to expand the emulation monitor's capabilities. Comment delimiters must be removed from some functions in the monitor before they can function. If you modify the emulation foreground monitor, the basic communication protocol between the emulation foreground monitor and the emulation software must be maintained. A detailed description of the communication protocol and the standard emulation monitor is given in this chapter.

---

## Comparison of Foreground and Background Monitors

An emulation monitor is required to service certain requests for information about the target system and the emulation processor. For example, when you request a register display, the emulation processor is forced into the monitor. The monitor code has the processor dump its registers into certain memory locations, which can then be read by the emulator system controller without further interference.

### Background Monitors

A *background* monitor is an emulation monitor which overlays the processor's memory space with a separate memory region. Entry into the monitor is accomplished by jamming the monitor addresses onto the processor's data bus.

Usually, a background monitor will be easier to work with in starting a new design. The monitor is immediately available upon powerup, and you don't have to worry about linking in the monitor code or allocating space for the monitor to use the emulator. No assumptions are made about the target system environment; therefore, you can test and debug hardware before any target system code has been written. All of the processor's address space is available for target system use, since the monitor memory is overlaid on processor memory, rather than subtracted from processor memory.

However, all background monitors sacrifice some level of support for the target system. For example, when the emulation processor enters the monitor to display registers, it will not respond to target system interrupt requests. This may pose serious problems for complex applications that rely on the microprocessor for real-time, non-intrusive support. Also, the background monitor code can't be modified to handle special conditions.

### Foreground Monitors

A *foreground* monitor may be required for more complex debugging and integration applications. A foreground monitor is a block of code that runs in the same memory space as your program. Foreground monitors allow the emulator to service real-time events, such as interrupts or watchdog timers, while executing in the monitor. For most multitasking, interrupt intensive applications, you will need to use a foreground monitor.

You can tailor the foreground monitor to meet your needs, such as servicing target system interrupts. However, the foreground monitor does use part of the processor's address space, which may cause problems in some target systems. You must also properly configure the emulator to use a foreground monitor (refer to chapter 4 "Answering Emulation Configuration Questions").

You may link the foreground monitor with your code. However, if possible, linking the monitor separately is preferred. This allows the monitor to be downloaded before the rest of your program. Linking monitor programs separately is more work initially, but it should prove worthwhile overall, since the monitor can then be loaded efficiently during the configuration process at the beginning of a session.

---

## **Using Both Foreground and Background Monitors in the HP 64430 Emulator**

Most conventional emulators are implemented with either a background or foreground monitor as the emulation control program. The emulator designer makes the appropriate compromises regarding the emulator's transparency during running the emulation monitor and picks one type of monitor or another in implementing the emulator.

Due to the on-chip MMU of the 68030 processor, however, and due to the full virtual system support provided by the HP 64430 emulator, both background and foreground monitors are provided and supported in the product. The decision by the user to enable either monitor is governed by the development stage and nature of the target system.

### **When to Use the Background Monitor**

It is recommended that background operation be chosen during the early stage of hardware development where full functionality of the target's interrupt, bus error, and other asynchronous events is not needed yet. The background monitor in that case has the advantage of being easy to use and can enable the emulator to be a stimulus for aiding in turning on the target hardware without



requiring the target system to be fully functional. For example, the display/modify target memory feature can be used to stimulate the target's memory interface and help the designer in troubleshooting any defects in that part of his circuitry. All the emulator would show while in background is the bus cycle referencing the target address. With the aid of an external timing analyzer (for example the HP 16500A), the user can monitor the target's signal behavior during that cycle and find any problem(s) the target might be having.

Another feature that is provided by the background monitor and not supported in foreground is the display/modify physical memory. The implementation of this function requires that the on\_chip MMU be temporarily turned off so that logical and physical addresses are identical. This is not possible during foreground operation since the foreground monitor is running as part of the virtual system.

## **When to Use the Foreground Monitor**

If the target system hardware is close to being completely functional, then foreground monitor operation is more desirable. The emulator runs in a more transparent mode than with background. Interrupts, bus errors, and all other exceptions can be handled by the target system software as if the emulator were not present. All emulation and analysis functions are available to the user. The monitor program customization allows the user to add, delete, or change the source code to fit the particular application. Messages relating to certain events can be added and displayed on the emulation terminal, and the target programs can jump to the monitor at any time. Display/modify of coprocessor registers can be achieved by adding the proper code to the monitor program.

In systems using the on-chip MMU of the 68030 processor, external memory in the target system and emulation memory are accessed as physical addresses. Since the emulation host communicates with the emulation monitor through the logical space, and due to the paging and swapping nature of the 68030 MMU, the requirement that the foreground monitor be mapped to emulation memory was waived. Hardware was added to the emulator to allow for the foreground monitor to be linked in the logical space with the rest of the target code where the eventual physical location is defined at run time. The special data area of the monitor, where the host communication happens, is located in a

special memory mapped to the untranslated CPU space of the processor. This makes it much easier for the foreground monitor to be installed and used.

## **Customization of the Monitor Programs**

The background monitor can not be customized by the user.

The source code for the foreground monitor is provided with the HP64430 product and can be modified to best fit the target application. Customizing the emulation foreground monitor is discussed in another section later in this chapter.

---

## **The Break Function And The Emulation Monitor**

The emulation break circuitry uses the NMI (INT7) resource of the processor to force the user program to be interrupted and the emulation monitor to be entered. A break can be generated for an illegal memory reference, a bus condition that the analysis card detects, a request by the emulation software, or from the keyboard.

---

## Emulation Monitor Description

### Note



---

This section applies to both foreground and background monitors except the user has no access to symbols or entry points in the background monitor.

---

The emulation monitor is made up of the following major sections.

- The processor exception vector look-up table.
- The entry points into the monitor.
- The emulation command scanner.
- The command execution modules.

Each of these sections is discussed in the following paragraphs.

### The Exception Vector Table

The emulation foreground monitor is entered through processor exceptions. The emulation monitor program contains pseudo instructions that load the vector table with the addresses of the emulation monitor exception handlers. The emulation monitor exception table defines 68030 exception vectors for the convenience of the user.

The emulation monitor program is shipped from the factory with all of the exception vectors (except RESET and MONITOR SINGLE STEP) contained in comment fields. This is done to allow you to supply the addresses for your own exception routines. If you have not written any exception handlers, you should remove the comment delimiters (\*) from those provided in the monitor. This enables the processor to use the exception vector lookup table provided with the monitor program.

If the user application has its own RESET handler, the reset vector in the monitor must be modified to point to the user reset handler.

Also, the reset-to-monitor function must be disabled. This is done by modifying the emulation configuration. You must answer no to the configuration question "Reset into the monitor?". See chapter 4 for detailed information.

## Note



---

The portion of the monitor defining the exception vector table is ORG'ed to 0H, and is not relocatable as is the rest of the monitor. When configuring the emulator, be sure to map the first block of memory (0H) to supervisor\_data emulation ram. Otherwise, locate the vector table in ROM in the target system. Refer to the section in this chapter titled "Loading the Emulation Monitor" for details on mapping the emulation monitor into memory.

---

## Emulation Monitor Entry Point Routines

The emulation monitor entry point routines provide input handler routines for the various entry paths. Six separate paths are defined for monitor entry. Each path is distinguished from the others by means of a unique ENTRY\_ID code which is stored upon entry into the monitor. The emulation monitor entry point routines are MONITOR\_ENTRY, SWBK\_ENTRY, JSR\_ENTRY, RESET\_ENTRY, and EXCEPTION\_ENTRY.

### Monitor\_entry

MONITOR\_ENTRY is the entry point into the emulation monitor for breaks from the user's program. On a break to MONITOR\_ENTRY, the 68030 PC and status register should be placed on the stack as is normally done when an exception occurs. On entry to the monitor, the processor's registers are saved and the interrupt mask is restored (if you have modified your monitor to enable this function). The emulation monitor then executes the command scanner routines.

### Swbk\_entry

SWBK\_ENTRY is the entry point into the emulation monitor when a software breakpoint (i.e., a BKPT instruction inserted in your code by the HP 64000-UX system) is processed.

### **Jsr\_entry**

JSR\_ENTRY (foreground monitor only) is the entry point into the emulation monitor that should be used if the user wants to jump directly to the emulation monitor. If running in supervisor mode, you can use the instruction "JSR JSR\_ENTRY" to jump to the emulation monitor. If the 68030 processor is running in user mode, a trap exception should be used. The trap vector should point to MONITOR\_ENTRY.

### **Reset\_entry**

RESET\_ENTRY is the entry point into the emulation monitor when the 68030 processes the reset exception. RESET\_ENTRY sets up a default stack and sets the processor's registers to default values.

### **Exception\_entry**

A set of exception entry points (foreground monitor only) are provided to give status messages for the ten exception vectors after reset. These exception vectors are provided for the convenience of the user and may be deleted or modified. For more information on the exception vector entry points, see the emulation foreground monitor source program and the section in this chapter entitled "Modifying The Exception Vector Table".

## **Emulation Command Scanner**

The emulation command scanner normally rests in an idle loop labeled MONITOR\_LOOP. The system global MONITOR\_CONTROL is repetitively examined. If bit 15 is set to zero, the idle loop is resumed. If bit 15 is set to one, a command is present and the program branches to the appropriate command routine.

Bit 15 of MONITOR\_CONTROL is set by the Host and cleared by the monitor program. The lower byte of MONITOR\_CONTROL contains a command number against which the command table is compared. If a match is found, a command entry point will be retrieved from the table and the command will be executed. If a match is not found, the program will return to the idle loop. The command is considered complete when bit 15 of MONITOR\_CONTROL is set to zero.

## **Emulation Command Execution Modules**

The Emulation Monitor command execution modules are ARE\_YOU\_THERE, EXIT\_MONITOR, COPY\_MEMORY, COPY\_ALT\_REG, MON\_ALT\_REGISTERS, SYNCH\_START\_ENABLE, SIM\_INT\_ENABLE, SIM\_INT\_DISABLE, and SIMULATED\_INTERRUPT.

### **Are\_you\_there**

ARE\_YOU\_THERE is used by the host (the HP 64000-UX) to determine whether the processor is executing in the monitor or in the target system code. It can also pass an ASCII message to be displayed on the host system status line.

### **Exit\_monitor**

EXIT\_MONITOR reloads the processor's register image and exits to the user's program.

### **Synch\_start\_enable**

SYNCH\_START\_ENABLE causes EXIT to be delayed. The monitor will loop waiting for an emulator status bit that indicates a synchronized start among multiple emulators has been received before EXIT is executed. This wait loop may be aborted by any command.

### **Copy\_memory**

COPY\_MEMORY moves data between the monitor parameter block areas and target system memory. This command is used to modify and display target system memory.

### **Copy\_alt\_reg**

COPY\_ALT\_REG reads from and writes to coprocessor registers.

### **Mon\_alt\_registers**

MON\_ALT\_REGISTERS is a jump table which contains the address offset of the coprocessor register load/unload routine for each of the 8 possible coprocessors.

The MON\_ALT\_REGISTERS table should be set up to contain the load routine names - the table start. Offsets from the start of the table are stored so the entries will fit in 16 bits.

### **Simint\_enable**

SIMINT\_ENABLE is a user defined simulated interrupt function which allows you to implement interrupt driven code on an emulator which is out of circuit. This command must set the local simulated interrupt enable flag TRUE and store SIM\_INTS\_TRUE at SIM\_INT\_CONTENTS to reenale simulated interrupts on exit. If simulated interrupts are not disabled on entry to the monitor, the break softkey will not work.

### **Simint\_disable**

SIMINT\_DISABLE is a user defined simulated interrupt function which allows you to disable interrupt driven code on an emulator which is out of circuit. This command must set the local simulated interrupt enable flag FALSE and store SIM\_INTS\_FALSE at SIM\_INT\_CONTENTS to keep simulated interrupts disabled on exit. If simulated interrupts are not disabled on entry to the monitor, the break softkey will not work.

### **Sim\_interrupt**

SIM\_INTERRUPT is a user defined simulated interrupt function which allows you to implement interrupt driven code on an emulator which is out of circuit. This command will typically cause a branch to your interrupt handler by way of a trap instruction. When the command is complete, the host processor expects the processor to be in the monitor.

---

## Customizing The Emulation Monitor

### Caution



---

#### **POSSIBLE LOSS OF WORK SESSION!**

**SYSTEM MAY BECOME UNUSABLE.** Your customized portion of the emulation monitor must not exit the monitor program. Exiting the monitor will cause the entire system to become unsynchronized and, therefore, unusable.

You should not make any changes to portions of the monitor other than those described in the following paragraphs. Changes in other sections of the monitor may cause some features to stop working due to modifications on the stack, or because the information that is passed to and from the various sections has been affected.

---

For most systems, the emulation foreground monitor supplied with your emulator enables all emulation features to operate. Some systems, however, require modification to the emulation monitor program in order to maximize the effectiveness of the emulator. For this reason, the source program for the monitor has been provided and is thoroughly commented. Each of the standard routines within the code has been described so that you can easily make your modifications.

---

### Caution



---

#### **POSSIBLE LOSS OF ORIGINAL MONITOR SOURCE PROGRAM!**

Do not modify original monitor source program. You should copy the 68030 emulation monitor source program to your subdirectory before making any modifications. Do not modify the copy supplied with your emulation system software. You should keep that copy as a backup.

---



If you haven't already done so, copy the emulation monitor to your subdirectory by executing the following command:

```
cp /usr/hp64000/monitor/mon_68030.s mon_68030.s
```

You must execute the command "**chmod 666 mon\_68030.s**" on the file before you modify it. It is shipped with "read-only" permissions.

You should now modify the copy in your subdirectory.

**Note**



---

After modifying the monitor, be sure to reassemble and relink the monitor program.

---

## Modifying The Exception Vector Table.

Find the following program block in the emulation monitor:

```
* ORG $000          0: reset
*   DC.L SP_TEMP
*   DC.L RESET_ENTRY
*
* ORG $008          2: bus error
*   DC.L EXCEPTION_ENTRY
*
* ORG $00C          3: address error
*   DC.L EXCEPTION_ENTRY
*
* ORG $010          4: illegal instruction
*   DC.L EXCEPTION_ENTRY
*
* ORG $014          5: divide by zero
*   DC.L EXCEPTION_ENTRY
*
* ORG $018          6: CHK instruction
*   DC.L EXCEPTION_ENTRY
*
* ORG $01C          7: TRAPV
*   DC.L EXCEPTION_ENTRY
*
* ORG $020          8: privilege violation
*   DC.L EXCEPTION_ENTRY
*
* ORG $024          9: monitor single-step entry
*   DC.L MONITOR_ENTRY
*
* ORG $024          9: trace
*   DC.L EXCEPTION_ENTRY
*
* ORG $028          10: "A" Line
*   DC.L EXCEPTION_ENTRY
*
* ORG $02C          11: "F" Line
*   DC.L EXCEPTION_ENTRY
*
* ORG $030          12: unassigned and reserved by Motorola
*   DC.L EXCEPTION_ENTRY
*
* ORG $034          13: coprocessor protocol violation
*   DC.L EXCEPTION_ENTRY
*
* ORG $038          14: stack frame format error
*   DC.L EXCEPTION_ENTRY
*
* ORG $03C          15: uninitialized interrupt
*   DC.L EXCEPTION_ENTRY
*
* ORG $040          16: unassigned and reserved by Motorola
*   DC.L EXCEPTION_ENTRY
*
* ... other unassigned reserved entries
*
* ORG $05C          23: unassigned and reserved by Motorola
*   DC.L EXCEPTION_ENTRY
*
* ORG $060          24: spurious interrupt
*   DC.L EXCEPTION_ENTRY
*
* ORG $064          25: interrupt level 1 autovector
*   DC.L EXCEPTION_ENTRY
*
* ORG $068          26: interrupt level 2 autovector
*   DC.L EXCEPTION_ENTRY
```

```

* ORG $06C          27: interrupt level 3 autovector
*   DC.L EXCEPTION_ENTRY
* ORG $070          28: interrupt level 4 autovector
*   DC.L EXCEPTION_ENTRY
* ORG $074          29: interrupt level 5 autovector
*   DC.L EXCEPTION_ENTRY
* ORG $078          30: interrupt level 6 autovector
*   DC.L EXCEPTION_ENTRY
* ORG $07C          31: interrupt level 7 autovector
*   DC.L EXCEPTION_ENTRY
* ORG $080          32: TRAP #0
*   DC.L EXCEPTION_ENTRY
* ... other TRAP #n entries
* ORG $0BC          47: TRAP #15
*   DC.L EXCEPTION_ENTRY
* ORG $0C0          48: floating point coprocessor unordered condition
*   DC.L EXCEPTION_ENTRY
* ORG $0C4          49: floating point coprocessor inexact result
*   DC.L EXCEPTION_ENTRY
* ORG $0C8          50: floating point coprocessor divide by zero
*   DC.L EXCEPTION_ENTRY
* ORG $0CC          51: floating point coprocessor underflow
*   DC.L EXCEPTION_ENTRY
* ORG $0D0          52: floating point coprocessor operand error
*   DC.L EXCEPTION_ENTRY
* ORG $0D4          53: floating point coprocessor overflow
*   DC.L EXCEPTION_ENTRY
* ORG $0D8          54: floating point coprocessor signaling Not a Number
*   DC.L EXCEPTION_ENTRY
* ORG $0DC          55: unassigned and reserved by Motorola
*   DC.L EXCEPTION_ENTRY
* ORG $0E0          56: PMMU configuration error
*   DC.L EXCEPTION_ENTRY
* ORG $0E4          57: PMMU illegal operation
*   DC.L EXCEPTION_ENTRY

```

Now, using your editor, remove the comment delimiters (\*) from the start of each line of code (except the second ORG \$24 statement) to make your program look as follows:

```

ORG $000          0: reset
  DC.L SP_TEMP
  DC.L RESET_ENTRY

ORG $008          2: bus error
  DC.L EXCEPTION_ENTRY

ORG $00C          3: address error
  DC.L EXCEPTION_ENTRY

ORG $010          4: illegal instruction
  DC.L EXCEPTION_ENTRY

```

ORG \$014                   5: divide by zero  
   DC.L EXCEPTION\_ENTRY  
 ORG \$018                   6: CHK instruction  
   DC.L EXCEPTION\_ENTRY  
 ORG \$01C                   7: TRAPV  
   DC.L EXCEPTION\_ENTRY  
 ORG \$020                   8: privilege violation  
   DC.L EXCEPTION\_ENTRY  
   ORG \$024                   9: monitor single-step entry  
     DC.L MONITOR\_ENTRY  
  
 ORG \$024                   9: trace  
   DC.L EXCEPTION\_ENTRY  
 ORG \$028                   10: "A" Line  
   DC.L EXCEPTION\_ENTRY  
 ORG \$02C                   11: "F" Line  
   DC.L EXCEPTION\_ENTRY  
 ORG \$030                   12: unassigned and reserved by Motorola  
   DC.L EXCEPTION\_ENTRY  
 ORG \$034                   13: coprocessor protocol violation  
   DC.L EXCEPTION\_ENTRY  
 ORG \$038                   14: stack frame format error  
   DC.L EXCEPTION\_ENTRY  
 ORG \$03C                   15: uninitialized interrupt  
   DC.L EXCEPTION\_ENTRY  
 ORG \$040                   16: unassigned and reserved by Motorola  
   DC.L EXCEPTION\_ENTRY  
 ... other unassigned reserved entries  
 ORG \$05C                   23: unassigned and reserved by Motorola  
   DC.L EXCEPTION\_ENTRY  
 ORG \$060                   24: spurious interrupt  
   DC.L EXCEPTION\_ENTRY  
 ORG \$064                   25: interrupt level 1 autovector  
   DC.L EXCEPTION\_ENTRY  
 ORG \$068                   26: interrupt level 2 autovector  
   DC.L EXCEPTION\_ENTRY  
 ORG \$06C                   27: interrupt level 3 autovector  
   DC.L EXCEPTION\_ENTRY  
 ORG \$070                   28: interrupt level 4 autovector  
   DC.L EXCEPTION\_ENTRY  
 ORG \$074                   29: interrupt level 5 autovector  
   DC.L EXCEPTION\_ENTRY  
 ORG \$078                   30: interrupt level 6 autovector  
   DC.L EXCEPTION\_ENTRY  
 ORG \$07C                   31: interrupt level 7 autovector  
   DC.L EXCEPTION\_ENTRY  
 ORG \$080                   32: TRAP #0  
   DC.L EXCEPTION\_ENTRY  
 ... other TRAP #n entries

```
ORG $0BC          47: TRAP #15
DC.L EXCEPTION_ENTRY
ORG $0C0          48: floating point coprocessor unordered condition
DC.L EXCEPTION_ENTRY
ORG $0C4          49: floating point coprocessor inexact result
DC.L EXCEPTION_ENTRY
ORG $0C8          50: floating point coprocessor divide by zero
DC.L EXCEPTION_ENTRY
ORG $0CC          51: floating point coprocessor underflow
DC.L EXCEPTION_ENTRY
ORG $0D0          52: floating point coprocessor operand error
DC.L EXCEPTION_ENTRY
ORG $0D4          53: floating point coprocessor overflow
DC.L EXCEPTION_ENTRY
ORG $0D8          54: floating point coprocessor signaling Not a Number
DC.L EXCEPTION_ENTRY
ORG $0DC          55: unassigned and reserved by Motorola
DC.L EXCEPTION_ENTRY
ORG $0E0          56: PMMU configuration error
DC.L EXCEPTION_ENTRY
ORG $0E4          57: PMMU illegal operation
DC.L EXCEPTION_ENTRY
```

End out of your edit session, making sure that you save your changes.

By removing the comment delimiters from this section of the monitor, you have made the exception vector table usable. The table provides all addresses that the monitor needs to operate.

## Continuing Target System Interrupts While In The Emulation Monitor

The processor interrupt mask can be restored to its prebreak value to enable target system interrupts while in the monitor. You must edit the monitor program if you want to enable the interrupts while running in the emulation monitor.

Under the `MONITOR_ENTRY` label, you will find a commented section that describes re-enabling the interrupts.

```
MONITOR_ENTRY
* return from exception if already in the monitor
  TAS      MONITOR_SEMAPHORE
  BPL.B    BREAK_OK

RTE

BREAK_OK
* block interrupts
  ORI.W    #BLOCK_INTERRUPTS<INT_MSK_SHIFT,SR
```

Comment the instruction `ORI.W #BLOCK_INTERRUPTS<INT_MSK_SHIFT,SR` to use the interrupts while in the monitor. Be sure to save your changes.

## Sending Messages From the User Program To the Emulator Display

### Note



---

This option is only available with the foreground monitor.

---

The `PUT_MONITOR_MSG` routine in the emulation monitor gives you a way to send messages to the display status line. In order to use this feature, you must do the following steps:

1. Define the message in your user code.
2. Set a trap vector to point to the `PUT_MONITOR_MSG` routine.

3. Initiate the appropriate trap. This will cause a "message breakpoint" and leave the processor running in the emulation monitor.
4. If you want to continue execution of your user program, your program should pop one long word off the stack to clean up the stack after the trap.

An example program implementing the "message breakpoint" is shown below.

```

*****
*
*   PUT_MONITOR_MSG is entered if you set up a trap vector
*   to point to it. The purpose of PUT_MONITOR_MSG is to send a
*   monitor message to the emulator, even if the request is not
*   in supervisor space.
*
*   The protocol for using PUT_MONITOR_MSG is as follows:
*
*       1) Set a TRAP #n vector to point to PUT_MONITOR_MSG.
*       2) Push the address of the message onto the stack.
*          The message must be in data space.
*       3) Initiate the appropriate trap. This will cause a
*          "message breakpoint", and leave the processor running
*          in the monitor.
*       4) If you continue the run, your program should pop
*          one long word off the stack to clean up.
*
*****
PUT_MONITOR_MSG
* return from exception if already in the monitor
    TAS     MONITOR_SEMAPHORE
    BPL.B   PUT_MON_MSG_OK
    RTE

PUT_MON_MSG_OK
* block interrupts
    ORI.W   #BLOCK_INTERRUPTS<INT_MSK_SHIFT,SR

* save registers
    MOVEM.L D0-D7/A0-A6,PREGS
    MOVEC   SFC,A0
    MOVE.L  A0,SFC_REG
    MOVEC   DFC,A0
    MOVE.L  A0,DFC_REG

* read emulator status register
    MOVEQ   #FC_CPU_SPACE,D0
    MOVEC   D0,SFC
    MOVEC   D0,DFC
    MOVES.L EMUL_STATUS,D0

* clear low in_monitor bit
    BCLR    #LINMON,D0
    MOVES.L D0,EMUL_STATUS

```

```
* if a supervisor space break (- 8 because memory reference BTST is a byte op)
BTST    #SUPRVISOR_STATE-8,(SP)
BEQ.B   USER_FRAME
```

```
PUT_MON_MSG_1
```

```
* put supervisor data function code into message parameter area
MOVE.L  #FC_SUPER_DATA,MON_MSG_FC

* save message address from below trap frame on stack
MOVE.L  (FOUR_WORD_SIZE*2,SP),MONITOR_MESSAGE
BRA.B   FINISH_MESSAGE
```

```
USER_FRAME
```

```
* else stack is in user data space
* put user data function code into message parameter area
MOVE.L  #FC_USER_DATA,MON_MSG_FC

* get user stack pointer
MOVE.L  #FC_USER_DATA,D0
MOVEC   D0,SFC
MOVE    USP,A0

* save message address from top of stack
MOVES.L (A0),D0
MOVE.L  D0,MONITOR_MESSAGE
```

```
FINISH_MESSAGE
```

```
* set message pending bit and set why_there to MON_MSG_RECVD
MOVEQ   #FC_CPU_SPACE,D0
MOVEC   D0,SFC
MOVEC   D0,DFC
MOVES.W MONITOR_CONTROL,D0
BSET    #MON_MSG_PEND,D0
MOVEQ   #MON_MSG_RECVD,D1
BFINS   D1,D0(WHY_THERE_START:WHY_THERE_WIDTH)
MOVES.W D0,MONITOR_CONTROL
BRA.W   MONITOR_MAIN
```

```
*****
```



---

# Emulation Monitor Memory Requirements For The 68030

The memory available in the emulation hardware is divided into 256-byte blocks of address space by the emulation system. Each 256-byte block begins on an even address multiple of 100H.

The relocatable program area of the emulation monitor requires approximately 3900 bytes of memory. You can determine this if you look at the **MODULE SUMMARY** section of the linker listing file (see below). You can see, in this example, that the emulation monitor begins at address 100H and ends at address 1023H. The program takes up 0A6B hexadecimal locations of memory. The value 0F23H is approximately 3900 decimal. Therefore, the emulation monitor can be mapped into 16 256-byte blocks of memory.

## MODULE SUMMARY

-----

MODULE	SECTION:START	SECTION:END	FILE
mon_68030	9:00000000	9:00000000	/hp/emul32/processor/m68030 /monitor/mon_68030.o
	mon_prog:00000100	mon_prog:00000B6B	
	mon_data:00000B6C	mon_data:00001023	
	:00000024	:00000027	

These memory requirements assume that the blocks each start on a 256-byte boundary and that the standard emulation monitor is being loaded. To check the memory requirements for the emulation monitor being used, the linker listing file should be checked.

The monitor program must reside in supervisor space. See the section "Loading The Emulation Monitor" in this chapter for details.

---

## Linking The Emulation Foreground Monitor

The emulation foreground monitor must be assembled and linked before it can be used by the emulation system. It can be linked with the target system code to produce one absolute file or it can be linked by itself.

---

## Loading The Emulation Monitor

The following rules must be followed when loading the emulation monitor:

1. Data space of the monitor must be mapped as RAM as opposed to ROM. The monitor transfer buffer, as well as many monitor "housekeeping" variables must be read and write accessible, and must, therefore be mapped as RAM.

In addition, portions of the monitor must write to other monitor program locations. Since writes to ROM are always blocked, the program section, as well as the data section, of the monitor must be mapped to RAM.

2. The emulation monitor is executed in response to a level 7 interrupt. Therefore, it is always executed within supervisor space and must be located in supervisor space. If the supervisor/user function code bit is not in use, this restriction does not apply.

The emulation software recognizes only program symbols. In the case of the monitor, the symbol addresses are assumed to be associated with the SUPR\_PROG function code (since the monitor is basically an interrupt routine). When the host writes control information to, or reads information from the monitor, it must use the special data space located in CPU space.

---

## Using Reset Into Foreground Monitor

If reset into the foreground monitor is specified as an option during emulation configuration (refer to chapter 4), some memory - either target or emulation - must be mapped to 0H SUPR\_PROG.

---

# Notes



# Using Custom Coprocessors

---

## Overview

This chapter provides the following information:

- A discussion of the requirements for using custom coprocessors.
- A detailed description of the custom coprocessor format file.
- A detailed description of how to modify the emulation monitor for use with custom coprocessors.
- A description of the emulation configuration questions related to custom coprocessors.



---

## Introduction


### Note



---

Custom register access is only supported with the foreground monitor enabled, except in the case of the MMU registers. MMU registers display and modification are supported for both foreground and background monitors.

---



The 68030 emulator has the capability to access floating point coprocessors and other coprocessors in your target system. You can both display and modify coprocessor register sets.

In order to use custom coprocessors with the emulator, you must:

- Provide a custom register format file defining the coprocessor address, size, and name and defining the register display format.
- Modify the emulation monitor program to include a storage buffer for the coprocessor registers, read/write routines to access coprocessor registers, and a pointer to the coprocessor read/write routines.
- Specify the custom register format file to the emulator during emulation configuration.

An example custom register format file is provided with your emulation software. This file is named:

`/usr/hp64000/inst/emul32/0410/0204/custom_spec.`

Read/write routines for the MMU are provided in the emulation monitor program.

---

## The Custom Register Format File

A custom register format file must specify the coprocessor you want to use with emulation. This file specifies:

- Which coprocessors should be used.
- Which coprocessor space the coprocessors should be located in.
- How large the register buffer should be for transfers.
- What the display should look like for each coprocessor.
- What register names there are for register modifies.

This file is read when the emulation configuration file is processed. The custom register format specification is fairly simple. For each coprocessor register set defined in the file, the following items must appear in the order specified:

1. the coprocessor address
2. the coprocessor size
3. the coprocessor name
4. the display spec

You may place comments in C language format (enclosed by `"/**` and `*/`) or blank lines before or after any register set, as well as between the specification fields. You can specify C language format include files using a control line of the form:

```
#include "filename"  
or  
#include <filename>
```

where the register set description could be placed in the include file. Filename must be the full pathname for the include file.

Using include files simplifies your custom register specification file and allows you to easily remove a register set from the specification file, if necessary.

A listing of a sample custom register specification file is shown in figure 8-1 at the end of this section. Figures 8-2 and 8-3 show how the same file could be written using a sample include file and include command lines.

## Address Specification

The address specification is of the form:

`ADDR=n`

where n is the coprocessor identification code that defines the coprocessor space. The address must be a number between 0 and 7, inclusive. If two register sets in the format file have the same address, only the last specified register set is used. The first register set is ignored. ADDR 0 is reserved for the MMU. The address specified for the "fpu" coprocessor must match the external FPU coprocessor identification code.

## Size Specification

The size specification is of the form:

`SIZE=n`

where n is the size (in bytes) of the register set transfer buffer. The transfer buffer is used to transfer the register contents between the emulation monitor and the host system. This number must be between 0 and 1020, inclusive.

## Name Specification

The coprocessor name specification is of the form:

`NAME="string"`

where string is a unique name for the coprocessor. If the name is not unique, any previous register specs with the same name will be ignored. The string must only contain alphanumeric characters. Register set names are available on softkeys during display, copy, and modify commands. Register set names are also placed in the header of the register display if the coprocessor set is active during the display.



## Register Set Display Specification

The register set display specification is enclosed by two lines as follows:

```
DISPLAY_START  
  <display specification>  
DISPLAY_END
```

The DISPLAY\_START and the DISPLAY\_END lines cannot have any trailing blanks. Any statements within these lines are used to generate the register display. These lines also provide information to the emulator for setting up register names for the modify command. Register specifications have the form:

```
NAME %OFFSET.WIDTH
```

where: NAME is the name that the register should be referenced by during display and modify commands.

OFFSET is the index into the register buffer (in bytes) to the location of the register contents.

WIDTH is the register width (in bytes).

All other text and white space in the register specification is presented in the display exactly as specified in the format file.

```

/*****/
/*                                     */
/*   COPROCESSOR DISPLAY FORMAT SPECIFICATIONS   */
/*                                     */
/*****/
/* This file contains the display format specifications for all coprocessors */
/* configured for this system. It may contain up to 7 other coprocessor */
/* specifications. */
/* */
/* The entry below describes the format for an 68882 fpu, and is used */
/* as an example. There are several pieces of data which MUST be supplied */
/* for each specification: */
/* */
/* ADDR=n, where n is in the range 0-7. This is the coprocessor id-code */
/* for the current entry. Please note that ADDR=0 is reserved for */
/* the MMU, and that all ADDR designations should appear only */
/* once in this file. */
/* */
/* SIZE=n, where 0 < n < 1020 bytes. SIZE describes the number of bytes */
/* in the monitor register buffer the user has defined for this */
/* coprocessor. */
/* */
/* NAME="string", where "string" is the UNIQUE name of the current */
/* coprocessor. The name is made up of alphanumeric characters */
/* only. This name will show up on a softkey when */
/* attempting to display/modify registers within emulation. */
/* */
/* DISPLAY_START marks the start of the display format spec for the */
/* current coprocessor. */
/* */
/* DISPLAY_END marks the end of the display spec, and also the end */
/* of the information for the current coprocessor. A new speci- */
/* fication may follow each DISPLAY_END. */
/* */
/* Within the bounds of DISPLAY_START and DISPLAY_END is the information */
/* needed to generate the display for each coprocessor. Each register */
/* description contains a name field and a register format field. The format */
/* field is in the form: */
/* */
/* %OFFSET.WIDTHr, where OFFSET is the index into the register buffer */
/* defined in the monitor (in bytes), and WIDTH is the width of */
/* the register (also in bytes). All other text, white space, */
/* etc, are preserved in the display. */

```

Figure 8-1. Sample Custom Register Specification File

```

/*****/
/*
/* EXAMPLE 68882 FPU SPECIFICATION */
/*
/*****/

ADDR=1      /* the fpu id-code (special: set by configuration) */
SIZE=108    /* number of bytes in the fpu register buffer */
NAME="fpu"  /* name of the fpu coprocessor (do not change) */

DISPLAY START
  FP0 %00.12r  FP1 %12.12r  FPCR %96.4r
  FP2 %24.12r  FP3 %36.12r  FPSR %100.4r
  FP4 %48.12r  FP5 %60.12r  FPIAR %104.4r
  FP6 %72.12r  FP7 %84.12r

DISPLAY_END

/* Other custom coprocessor display formats follow... */

```

**Figure 8-1. Sample Custom Register Spec. File (Cont'd)**

```

/*****/
/*
/* EXAMPLE 68882 FPU SPECIFICATION */
/*
/*****/

ADDR=1      /* the fpu id-code (special: set by configuration) */
SIZE=108    /* number of bytes in the fpu register buffer */
NAME="fpu"  /* name of the fpu coprocessor (do not change) */

DISPLAY START
  FP0 %00.12r  FP1 %12.12r  FPCR %96.4r
  FP2 %24.12r  FP3 %36.12r  FPSR %100.4r
  FP4 %48.12r  FP5 %60.12r  FPIAR %104.4r
  FP6 %72.12r  FP7 %84.12r

DISPLAY_END

```

**Figure 8-2. Custom Reg. Spec. Include File fpu\_spec**

```

/*****
/*  COPROCESSOR DISPLAY FORMAT SPECIFICATIONS  */
/*  */
/*****
/* This file contains the display format specifications for all coprocessors */
/* configured for this system.  It may contain up to 7 other coprocessor */
/* specifications.  */
/*  */
/* The entry below describes the format for an 68881 fpu, and is used */
/* as an example.  There are several pieces of data which MUST be supplied */
/* for each specification:  */
/*  */
/* ADDR=n, where n is in the range 0-7.  This is the coprocessor id-code */
/* for the current entry.  Please note that ADDR=0 is reserved for */
/* the MMU, and that all ADDR designations should appear only */
/* once in this file.  */
/*  */
/* SIZE=n, where 0 < n < 1020 bytes.  SIZE describes the number of bytes */
/* in the monitor register buffer the user has defined for this */
/* coprocessor.  */
/*  */
/* NAME="string", where "string" is the UNIQUE name of the current */
/* coprocessor.  The name is made up of alphanumeric characters */
/* only.  This name will show up on a softkey when */
/* attempting to display/modify registers within emulation.  */
/*  */
/* DISPLAY_START marks the start of the display format spec for the */
/* current coprocessor.  */
/*  */
/* DISPLAY_END marks the end of the display spec, and also the end */
/* of the information for the current coprocessor.  A new speci- */
/* fication may follow each DISPLAY_END.  */
/*  */
/*  */
/* Within the bounds of DISPLAY_START and DISPLAY_END is the information */
/* needed to generate the display for each coprocessor.  Each register */
/* description contains a name field and a register format field.  The format */
/* field is in the form:  */
/*  */
/* %OFFSET.WIDTHr, where OFFSET is the index into the register buffer */
/* defined in the monitor (in bytes), and WIDTH is the width of */
/* the register (also in bytes).  All other text, white space, */
/* etc, are preserved in the display.  */
#include "/users/em68030/custom_spec/fpu_spec"
.
.
.

```

Figure 8-3. Custom Reg. Spec. File Using Include Files

---

## Emulation Monitor Changes

In order to access coprocessor register sets, you must make some minor changes to the emulation monitor. You must declare a register buffer for storing the coprocessor register values, modify two table entries, and provide register buffer read/write routines for each coprocessor register set that the emulation monitor will access.

### Defining a Coprocessor Register Buffer

A coprocessor register buffer must be allocated in the emulation monitor for each custom coprocessor you use with the emulator. The emulator uses this buffer for storing register values read from or written to the custom coprocessor. An example buffer (MMU\_REGS) is provided with the emulation monitor program. This buffer is declared in the emulation monitor as follows:

```
MMU_REGS
SRP_REG          DC.L  0
                  DC.L  0
CRP_REG          DC.L  0
                  DC.L  0
TC_REG           DC.L  0
TT0_REG          DC.L  0
TT1_REG          DC.L  0
MMUSR_REG        DC.W  0
```

Locate this declaration in the emulation monitor program and insert your custom coprocessor register buffer declarations immediately following it in the emulation monitor. For example, if you are using an MC68882 coprocessor in your target system, you might add the following register buffer declaration:

```
FPU_882_REGS
FP_REG           DS.L  24
CONTROL_REG      DC.L  0
STATUS_REG       DC.L  0
IADDR_REG        DC.L  0
FPU_882_END
```

## Modifying The MON\_CPU\_REG- ISTERS Table

After declaring your register buffers, you need to modify the MON\_CPU\_REGISTERS table. This table has entries labeled "COPROC\_REG\_n", where n is the coprocessor identification number. The coprocessor identification numbers specified in the format file must have their corresponding table entry set to point to a buffer that will be used to transfer the register data to and from the monitor. These are the buffers that you declared in the previous section. The default MON\_CPU\_REGISTERS table provided with your emulation monitor is shown in the following listing:

```
MON_CPU_REGISTERS
COPROC_REG_0  DC.L  MMU_REGS
COPROC_REG_1  DC.L  0
COPROC_REG_2  DC.L  0
COPROC_REG_3  DC.L  0
COPROC_REG_4  DC.L  0
COPROC_REG_5  DC.L  0
COPROC_REG_6  DC.L  0
COPROC_REG_7  DC.L  0
```

For example, if you want to add an FPU in your target system at coprocessor address 1, you might want to modify the MON\_CPU\_REGISTERS table as follows:

```
MON_CPU_REGISTERS
COPROC_REG_0  DC.L  MMU_851_REGS
COPROC_REG_1  DC.L  FPU_882_REGS
COPROC_REG_2  DC.L  0
COPROC_REG_3  DC.L  0
COPROC_REG_4  DC.L  0
COPROC_REG_5  DC.L  0
COPROC_REG_6  DC.L  0
COPROC_REG_7  DC.L  0
```

## Modifying The MON\_ALT\_REGISTERS Table

The second table you must change is under the symbol "MON\_ALT\_REGISTERS". This table has entries labeled "COPROC\_LOAD\_n", where n is the coprocessor identification number. These entries point to a coprocessor's read/write routine. An example read/write routine (FPU\_881\_COPY) is provided in the emulation monitor for use with an external FPU. The default MON\_ALT\_REGISTERS table provided with your emulation monitor is shown in the following listing:

```
MON_ALT_REGISTERS

COPROC_LOAD_0    DC.W    MMU_COPY-MON_ALT_REGISTERS
COPROC_LOAD_1    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_1    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_2    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_3    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_4    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_5    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_6    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_7    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
```

If you want to use a FPU in your target system as in the previous example, you would modify the MON\_ALT\_BUFFER table as follows:

```
MON_ALT_REGISTERS

COPROC_LOAD_0    DC.W    MMU_COPY-MON_ALT_REGISTERS
COPROC_LOAD_1    DC.W    FPU_882_COPY-MON_ALT_REGISTERS
COPROC_LOAD_1    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_2    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_3    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_4    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_5    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_6    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
COPROC_LOAD_7    DC.W    INVALID_CP_ID-MON_ALT_REGISTERS
```

where FPU\_882\_COPY is the copy routine you have written for your FPU registers.

## Writing Coprocessor Copy Routines

The coprocessor copy routine must both read from and write to the coprocessor registers. If the emulation monitor symbol "MON\_COMMAND" contains the value "6", then the routine should perform a read into the register data buffer specified above. If the symbol = 7, the routine should write the register set using the values in the register data buffer.

An external FPU read/write routine (FPU\_882\_COPY) is shown in the following listing. The external FPU copy routine provides you with a good example of how to write your copy routine.

```

*****
*
* FPU 881_COPY is an example routine that transfers the FPU registers
* to/from the FPU_881_REGS data area. This code is commented out since
* there is no coprocessor in the emulator as shipped.
*
* FPU_881_COPY may be used as an example load/unload routine for
* other coprocessors.
*
* If this code is activated by uncommenting it, then an entry of the form
*
* COPROC_LOAD_1   DC.W       FPU_881_COPY-MON_ALT_REGISTERS
*
* should be placed in the MON_ALT_REGISTERS table above.
*
* The block that defines FPU_881_REGS in the data segment must also be
* uncommented and an entry placed in the appropriate COPROC_REG_n
* variable e. g.
*
* COPROC_REG_1   DC.L       FPU_881_REGS
*
*****
*FPU_881_COPY
*  CPI.W       #READ_ALT_REGISTERS,MON_COMMAND
*  BEQ.B       FPU_881_READ
*
*FPU_881_WRITE
** local copy of FPU data -- FPU
*  LEA        FPU_881_REGS,A0
*  FSAVE     -(SP)
*  FMOVEM.X  (A0)+,FP0-FP7
*  FMOVEM.L  (A0)+,CONTROL/STATUS/IADDR
*  FRESTORE  (SP)+
*  BRA.W     LOOP_REENTRY
*
*FPU_881_READ
** FPU -- local copy of FPU data
*  LEA        FPU_881_END,A0
*  FSAVE     -(SP)
*  FMOVEM.L  CONTROL/STATUS/IADDR,-(A0)
*  FMOVEM.X  FP0-FP7,-(A0)
*  FRESTORE  (SP)+
*  BRA.W     LOOP_REENTRY
*
*****
*
* Custom coprocessor register load/unload routines (if any) should
* be inserted into the monitor here. Please note that the default
* coprocessor id for the assembler is 1. In order for the assembler
* to generate the correct code for other ids, the assembler flag
* "FOPT ID=n", n=0-7, should be set appropriately.
*
*****

```



---

## Answering Emulation Coprocesor Configuration Questions

After modifying the emulation monitor, you must reassemble the emulation monitor and relink your emulation monitor with your user file.

The final step in setting up your 68030 emulator to use custom coprocessors is to answer the emulation configuration questions relating to custom coprocessors. In the default emulation configuration, you will be asked the question:

**Any custom registers?**

Answer **yes** to enable use of custom coprocessors.

If you answered "yes" to the above question, the next question will be:

**Name of custom register format file?**

Enter the full pathname of your custom register format file.

Complete the remainder of the emulation configuration questions and save your changes to a configuration file. You are now ready to run emulation using custom coprocessors.

A complete and detailed description of the emulation configuration questions is given in chapter 4.

---

## Notes

# Using Simulated I/O And Simulated Interrupts

---

## Overview

This chapter provides the following information:

- A description of how to configure simulated I/O, with a section on simulated I/O restrictions.
- A discussion of simulated interrupts which includes:
  - How simulated interrupts functions.
  - Simulated interrupts versus real interrupts.
  - Simulated interrupt configuration.
- A description of how to modify the monitor to use simulated interrupts.

## Note



---

Simulated I/O will work with either the foreground or the background monitor. Simulated interrupts will work only with the foreground monitor. When the MMU is enabled, all addresses used for the simulated I/O configuration must be mapped transparently. In a target system, it is expected that I/O space will be mapped transparently.

---

---

## Configuring Simulated I/O

The simulated I/O subsystem must be set up by answering a series of configuration questions. Your answers to these questions enable simulated I/O, set the control addresses, and define files used for standard I/O.

Detailed information on using simulated I/O with the emulator is provided in the *HP 64000-UX Simulated I/O Reference Manual*.

### Modify simulated I/O configuration? yes (no)

**no** Answering **no** causes the simulated I/O questions to be skipped. The current simulated I/O configuration is not modified.

**yes** Answering **yes** enables you to modify the simulated I/O configuration. The following questions are asked.

### Enable polling for simulated I/O? no (yes)

**no** Prevents the emulation software from reading the control address for simulated I/O commands. Answering **no** to this question enables you to disable simulated I/O while maintaining the current simulated I/O configuration. Later, when you need to enable simulated I/O, you can do so without having to re-enter control addresses or the file names for standard input, standard output, and standard error output. Answering **no** also causes the remaining simulated I/O questions to be skipped.

**yes** Causes the emulation software to frequently read the control address to determine if the user program has requested any simulated I/O commands. Answering **yes** causes the following questions to be asked.

### Function code data space? none (SUP\_DATA) (USR\_DATA)

This question asks you to specify the data space where the simio control addresses are located.

If during memory configuration, you specified **modify defined\_codes none**, you should use the default answer (**none**) here.

If you specified **modify defined\_codes all**, you should select **SUP\_DATA** or **USR\_DATA** as appropriate for your system.

If you specified **modify defined\_codes prog\_data**, you should select **USR\_DATA**.

**Simio control address 1? SIMIO\_CA\_ONE (<Addr>)**

**Simio control address 2? SIMIO\_CA\_TWO (<Addr>)**

**Simio control address 3? SIMIO\_CA\_THREE (<Addr>)**

**Simio control address 4? SIMIO\_CA\_FOUR (<Addr>)**

**Simio control address 5? SIMIO\_CA\_FIVE (<Addr>)**

**Simio control address 6? SIMIO\_CA\_SIX (<Addr>)**

The symbol **SIMIO\_CA\_ONE** is the default symbol associated with the first simulated I/O Control Address. The default symbol may be replaced with any valid symbol or an absolute address. If a symbol is specified, polling of that control address will not begin until a file containing that symbol is loaded. If an absolute address is specified, polling of that address will begin immediately.

The control address must be loaded into memory space assigned as RAM. User programs will run faster if the control address is located in emulation memory. Using target RAM causes the emulator to break into the monitor program every time the control address is polled for simulated I/O commands or data.

The following questions relate to the files associated with the three reserved file names "stdin", "stdout", and "stderr".

**File used for standard input? /dev/simio/keyboard (<FILE>)**

**File used for standard output? /dev/simio/display (<FILE>)**

**File used for standard error? /dev/simio/display (<FILE>)**

The default answers for these questions are "/dev/simio/keyboard", "/dev/simio/display", and "/dev/simio/display" respectively.

These files are not opened until Open (90H) is called with the file names "stdin", "stdout", and "stderr". These files are provided to allow easy redirection of input and output from the keyboard or display to a file or device without modifying the user program. (The compiler standard I/O libraries may open some or all of these reserved files automatically if simulated I/O is used. For more details, see the documentation on the simulated I/O libraries for the compiler you are using.)

## Restrictions On Simulated I/O

Restrictions on the use of simulated I/O are:

- There is a limit of 12 open files at any one time.
- There can only be four active simulated I/O processes at any one time.
- When using MMU, all simulated I/O control addresses must be mapped 1:1.
- When using MMU, the memory for simulated I/O must always be accessible from the supervisor state of the processor.

Since any simulated I/O file that is opened is associated with a file descriptor, opened files are independent of the control address. Up to 12 files can be opened with a single control address (CA). A total of six control addresses are allowed so that you can execute simulated I/O commands concurrently. Remember, a maximum of 12 simulated I/O files (between the six control addresses) may be open at any one time.

---

## Simulated Interrupts

Simulated interrupts enable you to test software which depends upon the occurrence of preemptive interrupts using out-of-circuit emulation. The simulated interrupt facility is enabled by writing a value of 0ffh to the simulated interrupt control address. The control address is defined during the emulation configuration session. The simulated interrupt facility, when enabled, generates approximately six interrupts per second, depending on what other emulation activities are occurring concurrently, i.e., simulated I/O and display updates.

The simulated interrupt facility can be used to test applications such as a preemptive scheduler in a multitasking system or interrupt driven I/O. Interrupt driven I/O can be simulated by executing simulated I/O commands when a simulated interrupt occurs.

An interrupt is a request by an external device that causes the processor to temporarily suspend normal execution in order to service the interrupting device. Normal execution resumes after the device has been serviced. Interrupts are asynchronous to normal execution. To simulate this action out of circuit, the emulation software running on the host system acts as the external device requesting service.

### How Does A Simulated Interrupt Function?

There are only two ways that the emulation software can interrupt the emulator. The first is to reset the processor in the emulator. Since a reset causes the current instruction counter to be lost, continuation of program execution is not possible. Therefore, reset is not usable for simulated interrupts. The second way to interrupt the emulator is to break to the monitor. This is the method used to implement simulated interrupts. Therefore, the emulation monitor must be loaded in order to use simulated interrupts.

The simulated interrupt begins when a value of 0ffh is written into the simulated interrupt control address. The emulation software polls this address just as it polls simulated I/O control addresses. When emulation finds the value 0ffh at the simulated interrupt control address, it causes a break to the emulation monitor.

The emulation monitor saves all registers as a normal part of the monitor entry sequence. The emulation monitor then loops, waiting for a command. The emulation software then sends a simulated interrupt command to the emulation monitor. The emulation monitor supplied with the emulator contains only a stub which immediately indicates completion.

However, a simulated interrupt is user definable. To create your own simulated interrupt, you must modify the emulation monitor. The modification must include the interrupt code needed to perform the action you want to happen when an interrupt occurs. Be aware of the time constraints discussed in the following section "Simulated Interrupts Versus Real Interrupts". A typical action is a TRAP instruction which vectors to your interrupt handler. See the example program given in figure 9-1. This feature is not available without modifying the monitor. For information on modifying the monitor for simulated interrupts, refer to the section of this chapter entitled "Modifying the Monitor to Use Simulated Interrupts".

Finally, after the interrupt is serviced, emulation sends the exit monitor command to the emulation monitor. The exit monitor command restores the registers that were saved upon entry to the monitor, which causes normal program execution to continue at the point where it was interrupted.

### **Simulated Interrupts Versus Real Interrupts**

There are some important differences between simulated interrupts and real interrupts. A simulated interrupt handler must return within a fixed amount of time. Part of the simulated interrupt configuration is the specification of the maximum amount of time that emulation should wait for an interrupt handler to complete execution. If the interrupt handler does not complete within the specified time, emulation forces a break to the monitor, reports a failure, and causes the simulated interrupt to terminate. It is not always possible to wait for simulated I/O to complete an interrupt handler.



```

*****
* This is a simulated interrupt test program. The vector for
* TRAP #14 is pointed to INT_HANDLER. The SIM_INTERRUPT command
* of the monitor must be modified to execute a TRAP #14. Notice
* that the SMIINT_CA is enabled, then a delay loop is executed,
* then SIMINT_CA is disabled. INT_HANDLER increments the location
* COUNTER to provide a count of the number of interrupts that occurred.
*
* NOTE
* Simulated interrupts must be enabled in the emulator configuration
* and the control address must be set to SIMINT_CA. To observe the
* number of interrupts occurring, use the following command:
*
* display memory COUNTER thru COUNTER+7 blocked long repetitively
*****

CHIP 68030

XDEF START,END1,INT_HANDLER
XDEF LOOP,SIMINT_CA,COUNTER

SECTION INTR_DATA

SIMINT_CA DC.L 0 ;Set up a memory location to
; be the control address.
COUNTER DC.L 0 ;Set up a memory location that
; the program writes to.

ORG 038H ;TRAP #14
DC.L INT_HANDLER ;Notice that the address of the
; interrupt handler routine is
; contained in the vector address
; for a TRAP #14.

SECTION INTR_PROG

START MOVE.L #0,COUNTER ;Clear the contents of the counter
; address.
MOVE.B #OFFH,SIMINT_CA ;Enable simulated interrupts.
MOVE.L #OFFFFFFFFH,D0 ;Set up a delay counter value.
LOOP SUBQ.L #1,D0 ;Delay for a while.
BNE LOOP
MOVE.B #0,SIMINT_CA ;Disable simulated interrupts now.
END1 BRA.B END1 ;Continuous loop.

INT_HANDLER ;This is the interrupt handler
; routine.
ADDQ.L #1,COUNTER ;Increment the contents of COUNTER.
RTE ;Return from exception.
END START ;Define the transfer address so that
; you may run or step from
; transfer_address.

```

Figure 9-1. Simulated Interrupt Test Program

While the emulation software may appear to be doing several things concurrently, for example; polling up to six simulated I/O control addresses, polling a simulated interrupt control address, and updating a display, it is in fact only a single HP-UX task performing each of these emulation tasks sequentially. This means that the simulated interrupt must complete before any of the other tasks can begin. That is a motivation for limiting the execution of a simulated interrupt handler to a very short period.

If the handler is permitted to execute for an indefinite period of time, it is possible for the entire emulation program to be 'locked up' by an interrupt handler that is waiting for an event that never occurs.

The final difference between simulated interrupts and real interrupts is that it is not possible for a simulated interrupt to occur while a simulated interrupt is being handled or while the emulator is executing in the monitor.

## Simulated Interrupt Configuration

The simulated interrupt facility is not available in real time mode. If real time mode is enabled, the simulated interrupt configuration questions are not asked. When real-time mode is not enabled, the command line displays the following question:

**Modify simulated interrupt configuration? no (yes)**

Press **Return** for the default (**no**) response.

Press **yes Return** to modify the simulated interrupt configuration.

If you answer **no**, the questions will be skipped. If you answer **yes**, the simulated interrupt questions will be asked. The simulated interrupt questions are:

**Enable polling for simulated interrupts? no (yes)**

**no**      if no is selected, emulation does not poll a simulated interrupt control address and never causes a simulated interrupt to occur.

**yes**     if yes is entered, the configuration questions are asked:

### Function code data space ? none (SUP\_DATA) (USR\_DATA)

This question asks you to specify the data space where the simulated interrupt control address is located.

If during memory configuration, you specified **modify defined\_codes none**, you should use the default answer (**none**) here.

If you specified **modify defined\_codes all**, you should select **SUP\_DATA** or **USR\_DATA** as appropriate for your system.

If you specified **modify defined\_codes prog\_data**, you should select **USR\_DATA**.

### Simulated interrupt control address? SIMINT\_CA (<Addr)

Enter the value of the simulated control address in response to this question. The value may be a symbolic value or a numeric value. The default is the symbolic value **SIMINT\_CA**.

If you are not linking the emulation monitor program with your target system program, you must be careful when using a symbolic control address such as **SIMINT\_CA**.

The monitor program will store the location of the control address each time that it executes. If you modify your program, and then reload the program without loading the monitor, there is a chance that the symbolic control address will have changed. The monitor program will not recognize a change unless you reload it.

If you do not reload the monitor each time that you load the target system program, you must **ORG** the control address to a specific location. If you **ORG** the address, make sure that you modify the "**Simulated interrupt control address**" configuration question to point to the new address.

Another solution is to link the monitor program with your program. This causes the monitor to recognize any new address because it loads with your program.

A similar consideration occurs if you modify the control address configuration question. If you are running your program, and then modify the configuration, you must reload your program (and the monitor). Otherwise, the system software does not recognize the new control address and may write to an unknown address.

**Maximum delay (in milliseconds) for simulated interrupt? 25  
(<NUMB>)**

The final simulated interrupt configuration question requests the time, in milliseconds, to allow a simulated interrupt handler to execute before assuming that execution of the handler has failed and generates a break to the monitor.

The default time is 25 milliseconds. The default time is approximately equal to the time required to initiate a simulated interrupt and check for its completion on an HP 9000. Even though the resolution of this specification is one millisecond, because of the time that is required to check for completion, the effective resolution is approximately 15 milliseconds. For example, changing the maximum delay from 25 milliseconds to 26 milliseconds probably has no effect on execution. Emulation does not always wait for the maximum delay to pass. If the interrupt handler completes any time before the maximum delay time, emulation forces an immediate return to the interrupted code.

The input to this question is limited to the range of 1 through 10000. Therefore, the maximum delay is 10 seconds. This upper limit was chosen to prevent 'locking up' emulation by an interrupt handler that fails to terminate.

If the user's interrupt handler routine exceeds the maximum delay allowed, the following error message appears on the status line: **"ERROR: Simulated interrupt failed to complete"**.

## **Restrictions On Simulated Interrupts**

Restrictions on the use of simulated interrupts are:

- Simulated interrupts are not supported by the background monitor.
- When using MMU, all simulated interrupt control addresses must be mapped 1:1.
- When using MMU, the memory for simulated interrupts must always be accessible from the supervisor state of the processor.

---

## Modifying The Monitor To Use Simulated Interrupts

The user defined simulated interrupt function allows you to implement interrupt driven code on an emulator which is out of circuit. This command will typically cause a branch to your interrupt handler by means of a TRAP instruction. This command must set the boolean variable `SIM_INTS_ENABLED` to `TRUE` and copy the control address to `SIM_INT_CA` so that the monitor can disable simulated interrupts on entry. If simulated interrupts are not disabled on entry to the monitor, the **break** softkey will not function.

The monitor program must be modified before you can use the simulated interrupt feature. Find the following block of code shown in figure 9-2 in the monitor program.

The TRAP #14 instruction will cause the interrupt routine to be serviced. You must uncomment the instruction or, if you wish to use a different instruction, you must provide the instruction in the same area of the monitor as the TRAP #14 instruction. If you use another TRAP or different instruction, you must be sure that the routine will be found by the monitor. For example, if you use the TRAP #14 instruction, you must make sure that the address information for your exception routine is in the vector table at address 038h.

When you are finished editing the emulation monitor, be sure to save your changes. It will be necessary to re-assemble and relink the monitor in order to use the simulated interrupts feature.

```

*****
*
*   COMMAND 9 ... USER DEFINED SIMULATED INTERRUPT FUNCTION
*
*   THE USER DEFINED SIMULATED INTERRUPT FUNCTION ALLOWS THE USER TO
*   IMPLEMENT INTERRUPT DRIVEN CODE ON AN EMULATOR WHICH IS OUT OF
*   CIRCUIT.  THIS COMMAND WILL TYPICALLY CAUSE A BRANCH TO THE USERS
*   INTERRUPT HANDLER VIA A TRAP INSTRUCTION.  THIS COMMAND MUST SET
*   THE BOOLEAN SIM_INTS_ENABLED TO TRUE AND COPY THE CONTROL ADDRESS
*   TO SIM_INT_CA SO THE MONITOR CAN DISABLE SIMULATED INTERRUPTS ON
*   ENTRY.  IF SIMULATED INTERRUPTS ARE NOT DISABLED ON ENTRY TO THE
*   MONITOR, THE break SOFTKEY WILL NOT WORK.
*
*   THE 64000 WILL SET UP MONITOR CMD_BUF; SCR_ADDR TO Simulated
*   interrupt control address and issue COMMAND 8009H.
*
*   WHEN THE COMMAND IS COMPLETE, THE 64000 EXPECTS THE PROCESSOR
*   TO BE IN MONITOR.
*
SIM_INTERRUPT
*   A NON-ZERO VALUE INDICATES THAT SIMULATED INTERRUPTS ARE ENABLED
  MOVE.B    #0FFH,SIM_INTS_ENABLED
*
*   STORE 0FFH AT SIM_INT_CONTENTS TO KEEP SIMULATED INTERRUPTS ENABLED
  MOVE.B    #0FFH,SIM_INT_CONTENTS
*
*   STORE THE INTERRUPT CONTROL ADDRESS THAT WAS PASSED BY THE 64000
  MOVE.L    SRC_ADDR,D0
  MOVE.L    D0,SIM_INT_CA
*
*   INSTRUCTIONS TO BRANCH TO THE USERS INTERRUPT HANDLER GO HERE
*   THIS WILL TYPICALLY BE A TRAP INSTRUCTION.
*
  TRAP     #14
*
  JMP      LOOP_REENTRY
*****

```

**Figure 9-2. Simulated Interrupt Function Code**



## How The Emulator Works

---

### Overview

This chapter describes how the following emulator functions work:

- The are\_you\_there monitor function.
- The run command.
- Software breakpoints.
- Single stepping with foreground monitor.
- Single stepping with background monitor.
- Target memory transfers.
- Displaying CPU registers.
- Modifying CPU registers.

---

## Introduction

The information provided in this chapter will give you a better understanding of how the emulator works and how the emulator interacts with your target system. This information, along with the information provided in chapter 6, Using the Emulator, should help you use the emulator more effectively and avoid problems that can occur when the emulator is used with a target system (in-circuit emulation mode).

### Note



---

The algorithms described apply to both background and foreground monitors unless otherwise specified.

---



---

## Are You There Function?

The "are\_you\_there" monitor function is the means by which the host computer determines whether or not the 68030 CPU is executing the monitor at a particular time. It is used primarily to display the "running" and "running in monitor" status line messages.

It also performs the important function of checking to see that a break request (level 7 interrupt) resulted in a successful entry to the monitor. The host computer issues break requests for all emulation functions requiring the use of the monitor. If the break fails, the host computer is unable to complete the user specified command, and displays a "cannot break into monitor" message.

The following algorithm describes how the **are\_you\_there** function works.

1. The host computer writes the value 8000h (bit 15 = 1) to the monitor data location **MONITOR\_CONTROL**.
2. If the emulation monitor is executing, and has completed a previous command, it executes an idle loop. In the idle loop, the monitor is waiting for a user command or for the host to make an "exit monitor" request.

If the idle loop is executing and **MONITOR\_CONTROL** is set to 8000h by the host, the monitor responds by clearing bit 15 (**MONITOR\_CONTROL** = 0), and returning to the idle loop.

If the 68030 CPU is executing in the user program, bit 15 is not cleared, leaving **MONITOR\_CONTROL** set to 8000h.

3. The host computer reads monitor data location **MONITOR\_CONTROL**.

If bit 15 of **MONITOR\_CONTROL** = 0, the monitor is executing.  
If bit 15 of **MONITOR\_CONTROL** = 1, the user program is executing.

---

## The Run Command

The run command starts execution of your user program. The command allows you to run from a specified address, run until a specified address is executed, or run from a start address until a specified address. The following algorithms describe how the run command is implemented.

### Run From Command

When you execute the command "**run from** {SUPERVISOR\_STATE | USER\_STATE} <address>", the following algorithm is executed.

1. The host computer initiates a break to the monitor (level 7 interrupt).
2. The host verifies that the 68030 CPU is executing in the emulation monitor. If the monitor is not executing, the error message "cannot break into monitor" is displayed.
3. The host modifies the monitor copy of the return address obtained on entry to the monitor from the level 7 interrupt. It sets the return address to the value specified in the run command.
4. The host modifies the monitor copy of the CPU status register obtained on entry to the monitor from the level 7 interrupt.
  - a. If the command specifies "SUPERVISOR\_STATE", the host sets the SUPERVISOR/USER bit to 1 (supervisor) so that the 68030 CPU will execute in supervisor mode on exit from the monitor.
  - b. If the command specifies "USER\_STATE", the host sets the SUPERVISOR/USER bit to 0 (user) so that the 68030 CPU will execute in user mode on exit from the monitor.
5. The host initiates a return (RTE) to the user program from the monitor by writing the "exit monitor" command (value 8001H) to monitor variable **MONITOR\_CONTROL**.

6. The host verifies that the 68030 CPU has exited the monitor. If the emulator monitor is still executing, the error message "monitor did not respond to exit request" is displayed.

## Run Until Command

When you execute the command "**run until** <address>", the following algorithm is executed.

1. The host computer initiates a break to the monitor (level 7 interrupt).
2. The host verifies that the 68030 CPU is executing in the emulation monitor. If the monitor is not executing, the error message "cannot break into monitor" is displayed.
3. The host computer reads the 16-bit word at <address> and saves it internally.
4. The host inserts a BKPT instruction at <address>. The breakpoint is marked internally as a one-shot breakpoint.
5. The host initiates a return (RTE) to the user program from the monitor by writing the "exit monitor" command (value 8001H) to MONITOR\_CONTROL.
6. The host verifies that the 68030 CPU has exited the monitor. If the emulator monitor is still executing, the error message "monitor did not respond to exit request" is displayed.

## Run From ... Until Command

When you execute the command "**run from** {SUPERVISOR\_STATE | USER\_STATE} <address1> **until** <address2>", the following algorithm is executed.

1. The host computer initiates a break to the monitor (level 7 interrupt).
2. The host verifies that the 68030 CPU is executing in the emulation monitor. If the monitor is not executing, the error message "cannot break into monitor" is displayed.

3. The host computer reads the 16-bit word at <address2> and saves it internally.
4. The host inserts a BKPT instruction at <address2>. The breakpoint is marked internally as a one-shot breakpoint.
5. The host modifies the monitor copy of the return address obtained on entry to the monitor from the level 7 interrupt. It sets the return address to the value <address1> specified in the run command.
6. The host modifies the monitor copy of the CPU status register obtained on entry to the monitor from the level 7 interrupt.
  - a. If the command specifies "SUPERVISOR\_STATE", the host sets the SUPERVISOR/USER bit to 1 (supervisor) so that the 68030 CPU will execute in supervisor mode on exit from the monitor.
  - b. If the command specifies "USER\_STATE", then the host sets the SUPERVISOR/USER bit to 0 (user) so that the 68030 CPU will execute in user mode on exit from the monitor.
7. The host initiates a return (RTE) to the user program from the monitor by writing the "exit monitor" command (value 8001H) to MONITOR\_CONTROL.
8. The host verifies that the 68030 CPU has exited the monitor. If the emulator monitor is still executing, the error message "monitor did not respond to exit request" is displayed.

---

## Software Breakpoints



The following sections describe how the software breakpoint function is implemented in the 68030 emulator. Software breakpoints enable you to enter software breaks into your user program as an aid in debugging your user software. Software breakpoints are also used in the implementation of the run until command.

### Note



---


When using the foreground monitor, the exception vector table is referenced only in the case of permanent breakpoints, which make use of the trace exception vector (VBR+24h). If one-shot breakpoints are working correctly, but permanent breakpoints fail, verify that the trace exception vector properly references the monitor (memory location **MONITOR\_ENTRY**).

---

### Setting A Software Breakpoint



When you execute the command "**modify sw\_breakpoint set {permanent | oneshot} <bkpt\_addr>**", the system executes the following algorithm.

1. The host computer initiates a break to the monitor (level 7 interrupt).
  2. The host computer detects that we actually got to the monitor, issuing an error message "cannot break into monitor" if not.
  3. The host gets the 16-bit word at <bkpt\_addr> and saves it in ORIG\_INST in host system memory.
  4. The host inserts the BKPT instruction at <bkpt\_addr>.
  5. The host initiates a return (RTE) to the user program from the monitor.
  6. Host verifies that the emulation monitor was exited, and issues an error message if not.
- 

## Executing A Software Breakpoint

When the 68030 CPU executes the BKPT instruction specified during emulation configuration, the following events occur:

1. Emulation circuitry detects the occurrence of a BKPT instruction and responds by jamming into the emulation monitor at **SWBK\_ENTRY**.

### Note



---

Only the BKPT instruction specified during emulator configuration is recognized by the emulator.

---

2. The host detects that a breakpoint was executed and issues the message "breakpoint hit at address XXXX."
3. The host restores the original instruction saved in **ORIG\_INST** to **<bkpt\_addr>**.
4. The emulation monitor enters the idle loop, waiting for a user command.

## Executing A Run Command After Executing A Software Breakpoint

When you specify a run command after executing a software breakpoint, the following events occur:

"run"

1. The host computer determines if the last BKPT instruction detected is permanent or one-shot.
2. If the breakpoint is one-shot, the emulation monitor returns (RTE) to the user program to begin execution at address **BKPT\_ADDR**.
3. If the breakpoint is permanent, the 68030 CPU is instructed to single-step the instruction at **BKPT\_ADDR** and return to the monitor.

4. The host computer reads the emulation monitor variable **MONITOR\_CONTROL** to verify that the emulator is executing the emulation monitor. If the emulator is not executing in the monitor, the message "cannot break into monitor" is displayed and the run command is aborted.
5. The host resets the breakpoint and returns (RTE) to the user program as described in steps 2 through 6 of the "Setting A Software Breakpoint" section.

### **"run from ADDR"**

1. The host computer determines if the last BKPT instruction executed was permanent or one shot.
2. If the breakpoint is oneshot, the emulation monitor returns\* (RTE) to the user program and begins execution at address ADDR.
3. If the breakpoint is permanent and the "run from" address is set equal to the breakpoint address BKPT\_ADDR, the 68030 CPU is instructed to single-step the instruction at BKPT\_ADDR and return to the emulation monitor.
4. The host resets the breakpoint as described in steps 2 through 4 of the "Setting A Software Breakpoint" section and then returns\* (RTE) to the user program. User program execution begins at ADDR.

\*The returns to the user program are accomplished by modifying the stack so that the RTE instruction in the monitor will return to address ADDR, rather than the address originally contained on the stack.

---

## Single Stepping With Foreground Monitor

The following algorithm describes how the single-stepping function is implemented in the foreground monitor. The single-step function uses the trace exception vector in the exception vector table. If this vector (VBR+24h) is set incorrectly, single stepping will fail.

When the user executes a step command, the following events occur:

1. The host computer initiates a break to the emulation monitor program by means of a level 7 interrupt.
2. The host computer reads the emulation monitor variable **MONITOR\_CONTROL** to verify that the emulator is executing the emulation monitor. If the emulator is not executing in the monitor, the message "cannot break into monitor" is displayed and the step command is aborted.
3. The host instructs the monitor to set the trace bits in the 68030 microprocessor status register (T1=1, T0=0). This enables the 68030 trace function.
4. If the user specified a "from <address>" the host sets the program counter value on the return stack to <address> so that, upon returning from the monitor to the user program, program execution will begin at <address>.
5. The host initiates a return (RTE) to the user program from the monitor.
6. The 68030 CPU executes a single instruction, and takes the trace exception which reenters the monitor at **MONITOR\_ENTRY**. Note that the trace exception vector (VBR+24h) must reference **MONITOR\_ENTRY** for this to function correctly.
7. The host verifies that the emulator is executing in the monitor as described in step 2.



8. The host instructs the monitor to clear the trace bits in the 68030 microprocessor status register (T1 = 0, T0 = 0). This disables the 68030 trace function.
9. The emulation monitor enters an idle loop, waiting for a user command.

---

## Single Stepping With Background Monitor

The following algorithm describes how the single-stepping function is implemented in the background monitor.

When the user executes a step command, the following events occur:

1. The host computer initiates a break to the emulation monitor program by means of a level 7 interrupt.
2. The host computer reads the emulation monitor variable **MONITOR\_CONTROL** to verify that the emulator is executing the emulation monitor. If the emulator is not executing in the monitor, the message "cannot break into monitor" is displayed and the step command is aborted.
3. The host puts the emulator in "single step" mode by setting a control bit (STEP) to 1.
4. If the user specified a "from <address>" the host sets the program counter value on the return stack to <address> so that, upon returning from the monitor to the user program, program execution will begin at <address>.
5. The host initiates a return (RTE) to the user program from the monitor.

6. The STEP bit, being set to 1, initiates a BREAK action after one instruction has been executed. This forces the CPU to reenter the monitor at MONITOR\_ENTRY.
7. The host verifies that the emulator is executing in the monitor as described in step 2.
8. The emulation monitor enters an idle loop, waiting for a user command.

---

## Target Memory Transfers

The following section describes the two modes the emulator uses to transfer data to and from target memory. In the automatic mode, the emulation monitor always attempts to longword align the transfer. Due to the dynamic bus sizing facility of the 68030, this alignment improves total transfer time with 8 and 16-bit memory systems, but is most effective with 32-bit memory systems. This algorithm can be tuned to meet specific target system requirements.

Alternately, the display/modidy command can be issued so that all transfers can be made in a "byte", "word", or "longword" mode.

The "auto" mode is described below:

1. At the beginning of the transfer, the monitor examines the lower two bits of the initial target system address to be read from or written to.
  - a. If bit 0 of this address is 1, the monitor transfers a single byte to or from the target system using a **MOVES.B** instruction. Following this, the target system address is incremented by one to reflect the next address to be transferred.
  - b. If bit 0 of the initial target system address is 0, the byte transfer and address increment does not occur.

This first step causes the target system address to be aligned to a word address, where bit 0 of the address is 0.

2. The monitor examines bit 1 of the target system address.
  - a. If bit 1 of this address is 1, the monitor transfers a single word to or from the target system using a **MOVES.W** instruction. Then, the target system address is incremented by two to reflect the next address to be transferred.
  - b. If bit 1 of the initial target system address is 0, the word transfer and address increment does not occur.

This step aligns the target system address to a longword address, where bits 1 and 0 of the address are 0.

3. The target system address is now longword aligned; that is, address bits 1 and 0 are both 0. The bulk of the transfer is then carried out using longword transfers. The operation of the transfer up to this point is summarized in figure 10-1.

<u>Starting Addr</u> <u>Bits 1 and 0</u>	<u>Transfer Description</u>
1    1	a. Copy a byte to longword align b. Increment target address by 1 c. Copy a longword d. Increment target address by 4 e. Repeat steps "c" and "d"
1    0	a. Copy a word to longword align b. Increment target address by 2 c. Copy a longword d. Increment target address by 4 e. Repeat steps "c" and "d"
0    1	a. Copy a byte to word align b. Increment target address by 1 c. Copy a word to longword align d. Increment target address by 2 e. Copy a longword f. Increment target address by 4 g. Repeat steps "e" and "f"
0    0	a. Copy a longword b. Increment target address by 4 c. Repeat steps "a" and "b"

**Figure 10-1. Target Memory Transfers in Automatic Mode**

4. After each longword transfer, the monitor examines the number of bytes remaining in the transfer. If the number is 0, the transfer is complete, and the monitor returns to the idle loop. If the number of bytes remaining to be copied is less than 4 prior to a longword transfer, longword transfers are no longer used, and control passes to monitor code that finishes up the remaining bytes (3, 2 or 1) of the transaction.
- a. If 3 bytes remain, a word transfer followed by a byte transfer is executed.
  - b. If 2 bytes remain, a single word transfer is performed.
  - c. If a single byte remains, a byte transfer is used. This monitor function is summarized in figure 10-2.

<u>Number of Bytes Remaining</u>	<u>Transfer Description</u>
4	a. Copy a longword b. Increment target address by 4 c. Return to monitor idle loop
3	a. Copy a word b. Increment target address by 2 c. Copy a byte d. Increment target address by 1 e. Return to monitor idle loop
2	a. Copy a word b. Increment target address by 2 c. Return to monitor idle loop
1	a. Copy a byte b. Increment target address by 1 c. Return to monitor idle loop
0	a. Return to monitor idle loop

**Figure 10-2. Monitor Operation At End Of Transfer**

## Displaying Target Memory

When you execute a display memory command with an address range mapped to target system memory, the emulation monitor reads the specified areas of target memory and copies the memory locations to an internal monitor buffer for transfer to the host computer. This process is described in the following steps:

1. The host computer initiates a break to the monitor (level 7 interrupt).
2. The emulation monitor enters the idle loop, waiting for a host command. The idle loop is located at monitor program symbol `MONITOR_LOOP`.
3. The host computer detects that the 68030 CPU is executing in the emulation monitor. If the CPU is not executing in the monitor, the host issues the error message "cannot break into monitor".
4. The host computer writes the memory transfer parameters to designated monitor locations listed as follows:

<b>Description</b>	<b>Monitor Location</b>
a. Number of bytes to read.....	PARM1
b. Starting address of target system read.....	PARM2
c. Function codes for target system read.....	PARM3
d. Starting address of monitor data buffer write.....	PARM4
e. Function codes for monitor data buffer write.....	PARM5
f. Access mode.....	PARM6

The monitor data buffer begins at monitor data symbol `MON_XFR_BUF` and is always referenced with the `CPU_SPACE` function code for the foreground monitor.

5. The host writes the "read user memory" command (8003H) to `MONITOR_CONTROL`. This causes the monitor to exit the idle loop and begin execution at monitor program symbol `COPY`.

6. The monitor sets up the transfer according to the six parameters listed above, and begins to copy target system memory values to the monitor data buffer using the algorithm described in the previous section. See the emulation monitor listing for additional details. Look at the monitor code following monitor program symbol **COPY**.
7. The host computer detects that the transfer has completed by observing a value of 0000H in **MONITOR\_CONTROL**. The host then reads and displays the information in the monitor data buffer. If the display memory command requested a display of more data bytes than the monitor transfer buffer can hold, the host computer sets up a new transfer for the remaining information by repeating the steps beginning with step 4.
8. The host computer initiates a return (RTE) to the user program from the monitor. This occurs as a result of the host writing the "exit monitor" command (8001H) to **MONITOR\_CONTROL**. This operation does not occur if the display memory command was issued while executing in the emulation monitor.

### **Copying from Target System Memory**

The algorithm for copying data from target memory is identical to that used when displaying target memory.

### **Modifying Target Memory**

When you execute a modify memory command with an address mapped to target system memory, the emulation monitor writes to the specified areas of target memory, copying data from the emulation monitor data buffer. The data in the emulation monitor buffer is put there by the host computer. The process for modifying target memory is described in the following steps:

1. The host computer initiates a break to the emulation monitor (a level 7 interrupt).
2. The monitor enters the idle loop, waiting for a command from the host computer. The idle loop is located at monitor program symbol **MONITOR\_LOOP**.

3. The host computer detects that the 68030 CPU is executing in the emulation monitor. If the CPU is not executing in the monitor, the host issues the error message "**cannot break into monitor**".
4. The host writes the memory transfer parameters to the designated monitor PARM1 through PARM6.
5. The host writes the "write user memory" command (8004H) to **MONITOR\_CONTROL**. This causes the monitor to exit the idle loop and begin execution at monitor program symbol **COPY**.
6. The monitor sets up the transfer according to the six parameters listed above, and begins to copy monitor data buffer values to the target system memory using the target memory transfer algorithm described previously. See the emulation monitor listing for additional details. Look at the monitor code following monitor program symbol **COPY**.
7. the host determines that the transfer has completed by observing a value of 0000H in **MONITOR\_CONTROL**. If the modify memory command requested a modify of more data bytes than could be held by the monitor transfer buffer, the host sets up a new transfer for the remaining information by repeating the steps beginning with step 4.
8. The host initiates a return (RTE) to the user program from the monitor. This results from the host writing the "exit monitor" command (8001H) to **MONITOR\_CONTROL**. This operation does not occur if the modify memory command was issued while executing in the emulation monitor.

### **Copying to Target System Memory**

The algorithm for copying data to target system memory is identical to that used when modifying target memory.

---

## Displaying The CPU Registers

When you execute a **display registers cpu** command, the following algorithm is executed:

1. The host computer initiates a break to the monitor (a level 7 interrupt).
2. The emulation monitor enters the idle loop, waiting for a command from the host computer. The idle loop is located at monitor program symbol **MONITOR\_LOOP**.
3. The host detects that the 68030 CPU is executing in the emulation monitor. If the CPU is not executing in the monitor, the host issues the error message "cannot break into monitor". The "are\_you\_there?" function is used to determine whether or not the monitor is executing.
4. The host reads and displays the register image save area that was constructed on entry into the monitor (i.e. the monitor data area starting with symbol **PCH** and ending with **DFCT**).
5. The host initiates a return (RTE) to the user program from the emulation monitor. This results from the host writing the "exit monitor" command (8001H) to **MONITOR\_CONTROL**. This operation does not occur if the **display registers cpu** command was issued while executing in the emulation monitor.



---

## Modifying The CPU Registers

When you execute a **modify registers cpu <regname> to <value>** command, the following algorithm is executed:

1. The host computer initiates a break to the emulation monitor (a level 7 interrupt).
2. The monitor enters the idle loop, waiting for a command from the host computer. The idle loop is located at monitor program symbol **MONITOR\_LOOP**.
3. The host detects that the 68030 CPU is executing in the monitor. If the CPU is not executing in the emulation monitor, the host issues the error message "cannot break into monitor". The "are\_you\_there?" function is used to determine whether or not the emulation monitor is executing.
4. The host writes the modified register value to the corresponding location in the register image save area constructed on entry to the monitor (i.e. the monitor data area starting with symbol **PCH** and ending with **DFCT**).
5. The host initiates a return (RTE) to the user program from the emulation monitor. This results from the host writing the "exit monitor" command (8001H) to **MONITOR\_CONTROL**. This operation does not occur if the modify registers cpu command was issued while the CPU was executing in the monitor.
6. When exiting the monitor, the register image save area is read to reload all CPU registers with their original values on initial entry to the monitor (see monitor program symbol **RTN3**). Since the modify registers command changes values in the register image save area, these new values are loaded in the CPU registers on exit from the monitor.

---

## Notes




## Emulation Error Messages

---

### 68030 Emulation Error Messages

This appendix contains a list of 68030 emulation error messages with descriptions of the error and information on how to correct the error, when appropriate. This list describes the most serious emulation errors that you may encounter.

Messages are listed in alphabetical order.



#### Attempt to read guarded memory, addr = XXXX

This message appears when an attempt is made to display a memory location mapped as "guarded" via the "display memory" command. The offending address is displayed in the XXXX field.

#### Attempt to write guarded memory, addr = XXXX

This message appears when an attempt is made to modify a memory location mapped as "guarded" via the "modify memory" command. The offending address is displayed in the XXXX field.

#### cannot break into monitor

This message is displayed when the host expects to find the CPU executing the monitor, but the "are\_you\_there?" function indicates otherwise. This message occurs after issuing a command that normally causes a break to the monitor.

If SUPERVISOR\_PROG and SUPERVISOR\_DATA areas are not overlaid for the emulation monitor, the "are\_you\_there?" function cannot function properly, resulting in this error message. If function codes are not in use, mapping overlays are not required.

To determine the cause of the failure, setup an analysis trace to trigger on the acknowledge cycle for the level 7 interrupt:

```
trace trigger_on a= 0ffffffh s= fcode CPU_SPACE
```

If the analyzer does not trigger, then it is likely that no level 7 interrupt was generated by the emulator. Check that the "Enable emulator use of INT7?" configuration question has been answered "yes". If so, a hardware error has occurred or the CPU is in a Reset, halt or DMA state (in which case the CPU will not respond to the level 7 interrupt in a timely manner).

The tracelist should show an emulator generated jam cycle. MONITOR\_ENTRY should be the address supplied by these cycles. Compare the tracelist of the monitor entry point to a monitor listing. Determine that the monitor has not been inadvertently overwritten. Be sure that the monitor area is overlaid with SUPERVISOR\_DATA and SUPERVISOR\_PROGRAM space (not necessary if function codes are turned off).

Check to see that the monitor enters, and stays in the monitor idle loop. If interrupts are enabled in the monitor, an external interrupt routine may be exiting the monitor and not returning properly. Or, if there are frequent interrupts being processed, the "are\_you\_there?" function may be simply timing out.

Next, setup the analyzer to trigger on the "are\_you\_there?" monitor command:

```
trace trigger_on a= MONITOR_CONTROL  
d= 8000xxxxH s= access READ
```

The address and data specifications may differ, depending on the address of MONITOR\_CONTROL, and the width of the memory system being referenced.

Determine that the "are\_you\_there?" function in the monitor (ARE\_THERE) is functioning properly by observing the trace after capturing the condition where MONITOR\_CONTROL is read as 8000H. Compare this trace to the monitor listing.

### **Could not disable breakpoint at address XXXX**

This message normally results from attempting to clear a breakpoint in target system memory, but for some reason, the emulator could not break into the monitor in order to clear the breakpoint.

**Could not enable  
breakpoint at  
address XXXX**

This message normally results from attempting to set a breakpoint in target system memory, but for some reason, the emulator could not break into the monitor in order to set the breakpoint. This message also occurs when attempting to set a breakpoint in target ROM, but does not occur when setting a breakpoint in emulation RAM or ROM. Trying to set a breakpoint in a guarded area of memory will also result in this error message.

**monitor did not  
respond to exit  
request**

This message is displayed when the host expects to find the CPU executing somewhere other than in the monitor, but the "are\_you\_there" monitor function indicates otherwise. This message occurs after issuing a command that results in a return to the user program from the monitor (i.e. display registers while the user program is executing, or "run" while in the monitor, etc.).

If SUPERVISOR\_PROG and SUPERVISOR\_DATA areas are not overlayed for the emulation monitor, the "are\_you\_there?" function cannot function properly, resulting in this error message. If function codes are not in use, mapping overlays are not required.

To determine the cause of the failure, setup an analysis trace to trigger on the "exit monitor" command. This can be done with the following trace specification:

```
trace trigger_on a= MONITOR_CONTROL  
d= 8001xxxxH s= access READ
```

Note that the address and data specifications may differ, depending on the address of MONITOR\_CONTROL, and the width of the memory system being referenced.

If the monitor is not executing (i.e. in an interrupt routine or elsewhere) at the time of an "exit monitor" command, the command cannot be recognized and this error message will result after a timeout.

Observe the exit mechanism from the monitor, and compare the acquired trace to the monitor listing. Be certain that the monitor has not been overwritten inadvertently.

Once the monitor is exited, check that the user program executes properly. If the user program returns to the monitor immediately after the "exit monitor" command is issued, this message appears.

**No breakpoint exists  
at address XXXX**

This message is emitted if the user attempts to clear a breakpoint at an address for which no breakpoint was previously specified. The emulation system is only aware of breakpoints set by the "modify sw\_breakpoints set ..." command. If a "modify memory ..." command was used to set the breakpoint, or if the breakpoint existed in the absolute code loaded into the emulator, it is not possible to clear such breakpoints using "modify sw\_breakpoints clear ..." commands.

**(no termination)  
message in tracelist**

This message normally indicates that a particular CPU cycle was terminated by LBERR or LHALT instead of the usual termination by DSACKs or STERM.

This message can also be a clue that the target system is violating the MC68030 specification which specifies that the DSACK signals must not be negated before address strobe is negated by the CPU. This is the case because the analyzer uses a derivative of address strobe as an analysis clock. If DSACKs are high prior to the low-to-high transition of address strobe, a "no DSACK" message can result.

**no memory cycles**

This status line message indicates that the emulator has not received a low-to-high or high-to-low transition on address strobe for at least 25-30 ms. This message most often appears when executing from cache, if there are no external cycles for long periods of time.

Any device that drives address strobe will inhibit the message, including the emulator 68030, DMA devices, and coprocessors. If a DMA mechanism, for example does not drive address strobe, this message may appear after the specified timeout. (Note that bus cycles where address strobe is not driven cannot be captured by the analyzer.)

This message is simply a warning that address strobes are infrequent.

**Reset (with capital  
"R")**

This message indicates that the CPU is being reset due to the use of the reset softkey in the emulation software.

**reset (with lower case  
"r")**

This message indicates that the CPU is being reset by target system hardware.

**running**

This message indicates the emulator is running in a user program.

**running in monitor**

This message indicates the emulator is running in the monitor.

**slow dev at a= XXXX  
(YY)**

This status line message indicates that the CPU is presently attempting to run a bus cycle, but the cycle has not completed after approximately 25 ms. This means that although the CPU asserted address strobe (set it low), the addressed memory (I/O device, etc.) has not yet returned DSACKs, STERM, BERR, and/or HALT as appropriate.

The XXXX field above indicates the address of the attempted cycle, and the YY field indicates the function code applied to the cycle according to the following table:

SD = Supervisor Data  
SP = Supervisor Program  
UD = User Data  
UP = User Program  
R0 = Reserved Address Space 0  
R3 = Reserved Address Space 3  
R4 = Reserved Address Space 4  
CS = CPU Space

Note that this message is simply a warning that the current cycle is taking an unusually long time to complete.

---

## Notes





# **Source Files For Getting Started Examples**

---

## **Introduction**

This appendix contains the listings of the "towers.c" and "simint.c" source files. These two source files were compiled and linked with the emulation monitor program to form the towers.X absolute file which was used in all the examples in this manual that show the internal analyzer making trace measurements. The towers.c source listing is first in this appendix. The simint.c source listing is last.

## Source File For towers.c

```
/* LSD:@(#) 0.06 88/06/16
/* @(mktid) (02.10 24Jun88)
/*
/* This program demonstrates the solution to the popular
/* "Towers of Hanoi" brain teaser puzzle. The puzzle consists
/* of 3 pegs and a number of discs of different diameters which
/* fit over the pegs. The discs are ordered by their diameter,
/* largest on the bottom, on one peg. The object is to move
/* all of the discs from one peg to another such that they end
/* up in the same order on the new peg using the minimum number
/* of moves. Only one disc can be moved at a time, and a larger
/* disc may never be placed on top of a smaller disc.
/*
/* The solution can be visualized using "display simulated io"
/* command. The number of discs is selected by responding to
/* the input request using the "modify keyboard to simio"
/* command and entering a number between 1 and 7. Multiple
/* numbers separated by spaces can be entered before hitting
/* return to get multiple executions of the program, and "C"
/* may be entered to run the program continuously.
/*
/* The speed of the program can be modified in real time with
/* the variable loc_delay and the "modify memory" command.
/*
/* NOTE: This file has been designed with the use of "ifdef"
/* to allow it to be compiled and run on the host as well as
/* cross compiled for the emulator.
/*
/* If not the AxLS 68030 C compiler, then probably not an ANSI
/* compiler so we must remove the const and void keywords.
#ifdef __m68030
#define const
#define void char
#endif

#include <stdio.h>
#include <stdlib.h>

#define TRUE 1
#define FALSE 0
#define NOVALIDENTRY 1
#define STDOUT 1
#define FIRSTCOL 0
#define LEFT 0
#define MIDDLE 1
#define RIGHT 2

#define MAX_DISC 7
#define MAX_CHARS 16
#define MAX_TOWERS 3
#define REPEAT 99
```



```

extern void enable_int();
extern void disable_int();
extern int sim_ints_serviced;
extern int sim_int_ca;
#pragma SECTION UNDO
#endif

/* for local forward referencing */
static void pause();
static void show_discs();
static void init_display();
static void towers();
static int ask_for_number();

main()
{
#ifdef INTERRUPTS
    enable_int();
#endif

    run_continuous = FALSE;

    while ( ask_for_number(&num_discs) == TRUE ) {
#ifdef __m68030
        clear_screen(STDOUT);
#endif
        move_num = 0;
        init_display(LEFT);
        show_discs();
        pause();

        towers(num_discs,LEFT,RIGHT,MIDDLE);

        show_discs();
    }

/* ANSI supports concatenated strings, AxLS simio has cursor control */
#ifdef __m68030
    pos_cursor(STDOUT, FIRSTCOL, num_discs+5);
    printf("\t\tPuzzle with %d discs can be solved in "
           "%d moves.      \n",num_discs,move_num);
#else
    printf("\n\t\tPuzzle with %d discs can", num_discs);
    printf(" be solved in %d moves.      \n", move_num);
#endif

    pause();
    pause();
}

return(0);
}

/***** Towers routines *****/

static int ask_for_number(num)
int *num;
{
    char err_char1,err_char2;
    int last_num,ret_val;

    last_num = *num;

```

```

        if ( run_continuous == FALSE ) {

#ifdef __m68030
        clear_screen(STDOUT);
        printf("\n\nExecute 'modify keyboard_to_simio' then enter"
              " one of the following:"
              "\n\tNumber of discs to use [1-%d]"
              "\n\t'0' to exit program"
              "\n\t'C' to run continuously using last number entered\n\n"
              ,MAX_DISC);
#else
        printf("\n\nEnter one of the following:");
        printf("\n\tNumber of discs to use [1-%d]",MAX_DISC);
        printf("\n\t'0' to exit program");
        printf("\n\t'C' to run continuously using last number entered\n\n");
#endif

        while ( NOVALIDENTRY ) {

            printf("?");
            /* scanf will return one of the following three values: */
            /* 1) "0" indicates a scanning error */
            /* 2) "1" indicates valid input */
            /* 3) "EOF" */

            ret_val = scanf("%d",num);

            switch (ret_val) {
                case 0:
                    err_char1 = getchar();
                    err_char2 = getchar();

                    /* If a "C" is entered, the last number will be
                     * used forever (if it was valid). */

                    if (err_char1 == 'C') {
                        if ( (last_num < 1) || (last_num >
                            MAX_DISC) )
                            puts(" invalid repeat
                                number!");
                        else {
                            *num = last_num;
                            run_continuous = TRUE;
                            return(TRUE);
                        }
                    }

                    /* If the user hits the "suspend" softkey, as a
                     * courtesy the emulator sends an escape 1
                     * character sequence allowing the program to
                     * detect that the input was suspended.
                     * else if ( (err_char1 == '\033') && (err_char2
                     * puts(" input suspended!");
                     * else puts(" input error, re-enter line!");

```

```

        fflush(stdin);
        /* try again! */
        break;

    case 1:
        if (*num == 0) {
            printf(" exiting!\n");
            return(FALSE);
        }
        else if ((*num < 0) || (*num > MAX_DISC))
            printf(" %d is not a valid
number!\n",*num);

            /* try again! */
        else
            return(TRUE);
        break;

    case EOF:
        return(FALSE);
        break;

        } /* end switch ret_val */

    } /* while no valid entry */

} /* if not run continuous */

return(TRUE);
}

static void pause()
{
    int i,j;

    for (i = 0; i < loc_delay; i++)
        for (j = 0; j < loc_delay; j++) {
            }
}

#ifdef DEBUGG
    rts_prefetch = 0;
#endif
}

static void show_discs()
{
    DISC *disc_ptr;
    char *string, *p;
    int disc, tower, ch;

    if ( ( p = string = (char *) malloc(BUFSIZ) ) == NULL ) {
        fputs("malloc failed\n",stderr);
        exit(1);
    }
}

#ifdef __m68030
    pos_cursor(STDOUT, FIRSTCOL, 1);
#endif

for (disc = 0; disc < num_discs; disc++) {
    for (tower = 0; tower < MAX_TOWERS; tower++) {
        *p++ = '\t';
        disc_ptr = &display[tower][disc];
        for (ch = 0; ch < MAX_CHARS; ch++)

```



```

#ifndef DEBUGG
    rts_prefetch = 0;
#endif
}

static void init_display(start_tower)
int start_tower;
{
    int tower,disc;

    /* initialize the display array to be blank */

    for (tower = 0; tower < MAX_TOWERS; tower++) {
        for (disc = 0; disc < MAX_DISC; disc++)
            display[tower][disc] = blank_disc;
        free_level[tower] = num_discs - 1;
    }

    /* place num_discs on the specified tower */
    for (disc = 0; disc < num_discs; disc++) {
        display[start_tower][disc] = disc_word[disc];
        disc_level[disc] = disc;
    }
    free_level[start_tower] = 0;
}

static void towers(n,from_peg,to_peg,aux_peg)
register int n,from_peg,to_peg,aux_peg;
{
    if (n == 1)
        move_disc(1,from_peg,to_peg);
    else {
        towers(n-1,from_peg,aux_peg,to_peg);
        move_disc(n,from_peg,to_peg);
        towers(n-1,aux_peg,to_peg,from_peg);
    }
}

```



---

## Source File For simint.c

```
/* LSD:@(#) .....
/* @(mktid) .....
/*
/* This file contains some very simple examples of routines to
/* use with the simulated interrupt mechanism of the emulator.
/*
/*
/* If not the AxLS 68030 C compiler, then probably not an ANSI
/* compiler so we must remove the const and void keywords.
#ifdef __m68030
#pragma SECTION PROG=simint DATA=data CONST=simint
#else
#define volatile
#define void char
#endif

/* This variable records the number of serviced simulated interrupts. */

int sim_ints_serviced = 0;

/* This variable will be used to control simulated interrupts and is
/* specified in the emulator configuration file. The host and the
/* monitor watch this "control address" to decide whether to perform
/* the interrupt function or not. Simulated interrupts will be
/* enabled when the control address flag is set to -1 and disabled
/* if it is set to 0. The "modify memory" command can be used to
/* enable and disable interrupts in real time once the program has
/* has been started. */

/* Remember that using the "volatile" keyword restricts the compiler's
/* optimization for the entire file, but guarantees proper access of
/* variable. */

volatile int sim_int_ca = -1;

void enable_int()
{
    /* enable simulated interrupts from emulator */
    sim_int_ca = -1;
}

void disable_int()
{
    /* disable simulated interrupts from emulator */
    sim_int_ca = 0;
}

#ifdef __m68030
#pragma INTERRUPT
static void sim_int_handler()
{
    /* service simulated interrupts from emulator */
    sim_ints_serviced++;
}
#endif
```

```
}  
  
/* Initialize the interrupt vector table to point to our routine. */  
#pragma SECTION DATA=0xb8  
void (*trap14)() = sim_int_handler;  
#pragma SECTION UNDO  
#endif
```

# Timing Comparisons

---

## Introduction

This appendix contains tables which list:

- Timing comparisons between the MC68030 and the HP 64430 emulator.
- DC electrical specifications for the HP 64430.

## MC68030/HP 64430 Timing Comparisons

### 13.5 AC ELECTRICAL SPECIFICATIONS -- CLOCK INPUT

Num	Characteristic	33.33 MHz <sup>15</sup>		HP 64430		Unit
		Min	Max	Min	Max	
	Frequency of Operation	20	33	20	33	Mhz
1	Cycle Time	30	80	30	80	ns
2,3	Clock Pulse Width	14	66	14	66	ns
4,5	Rise and Fall Times	---	3	---	3	ns

MC68030 electrical specifications reprinted courtesy Motorola, Inc.

### 13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES

(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; T<sub>A</sub> = 0° to 70° C)

Num	Characteristic	33.33 MHz <sup>15</sup>		HP 64430		Unit
		Min	Max	Min	Max	
6	Clock High to $\overline{FC}$ , $\overline{Size}$ , $\overline{RMC}$ , $\overline{CIOUT}$ Address Valid	0	14	0	14	ns
	Clock High to $\overline{IPEND}$ Valid	0	14	0	24	ns
6A	Clock High to $\overline{ECS}$ , $\overline{OCS}$ Asserted	0	12	0	15	ns
6B	$\overline{FC}$ , $\overline{Size}$ , $\overline{RMC}$ , $\overline{CIOUT}$ Address Valid to Negating $\overline{ECS}$	3	---	3	---	ns
	$\overline{IPEND}$ Valid to Negating $\overline{ECS}$	3	---	-1	---	ns
7	Clock High to $\overline{FC}$ , $\overline{Size}$ , $\overline{RMC}$ , $\overline{CIOUT}$ , Address, Data High Impedance	0	30	0	30	ns

### 13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; TA = 0° to 70° C)

Num	Characteristic	33.33 MHz <sup>15</sup>		HP 64430		Unit
		Min	Max	Min	Max	
8	Clock High to $\overline{FC}$ , $\overline{Size}$ , $\overline{RMC}$ , $\overline{IPEND}$ , $\overline{CIOUT}$ , Address Invalid	0	---	0	---	ns
9	Clock Low to $\overline{AS}$ , $\overline{DS}$ , $\overline{CBREQ}$ Asserted	2	10	2	15	ns
9A <sup>1</sup>	$\overline{AS}$ to $\overline{DS}$ Assertion Skew (Read)	-8	8	-8	8	ns
9B <sup>14</sup>	$\overline{AS}$ Asserted to $\overline{DS}$ Asserted (Write)	22	---	22	---	ns
10	$\overline{ECS}$ Width Asserted	8	---	7	---	ns
10A	$\overline{OCS}$ Width Asserted	8	---	7	---	ns
10B <sup>7</sup>	$\overline{ECS}$ , $\overline{OCS}$ Width Negated	5	---	5	---	ns
11	$\overline{FC}$ , $\overline{Size}$ , $\overline{RMC}$ , $\overline{CIOUT}$ , Address Valid to $\overline{AS}$ Asserted (and $\overline{DS}$ Asserted, Read)	5	---	5	---	ns
	$\overline{IPEND}$ to $\overline{AS}$ Asserted (and $\overline{DS}$ Asserted, Read)	5	---	-1	---	ns
12	Clock Low to $\overline{AS}$ , $\overline{DS}$ , $\overline{CBREQ}$ Negated	0	10	0	15	ns
12A	Clock Low to $\overline{ECS/OCS}$ Negated	0	15	0	16	ns
13	$\overline{AS}$ , $\overline{DS}$ Negated to $\overline{FC}$ , $\overline{Size}$ , $\overline{RMC}$ , $\overline{CIOUT}$ , Address Invalid	5	---	5	---	ns
14	$\overline{AS}$ (and $\overline{DS}$ , Read) Width Asserted (Asynchronous Cycle)	45	---	43	---	ns
14A <sup>11</sup>	$\overline{DS}$ Width Asserted, Write	23	---	21	---	ns
14B	$\overline{AS}$ (and $\overline{DS}$ Read) Width Asserted (Synchronous Cycle)	23	---	21	---	ns

### 13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; TA = 0° to 70° C)

Num	Characteristic	33.33 MHz <sup>15</sup>		HP 64430		Unit
		Min	Max	Min	Max	
15	$\overline{AS}$ , $\overline{DS}$ Width Negated	23	---	21	---	ns
15A <sup>8</sup>	$\overline{DS}$ Negated to $\overline{AS}$ Asserted	18	---	18	---	ns
16	Clock High to $\overline{AS}$ , $\overline{DS}$ , $R/\overline{W}$ , $\overline{DBEN}$ , $\overline{CBREQ}$ High Impedance	---	30	---	30	ns
17	$\overline{AS}$ , $\overline{DS}$ Negated to $R/\overline{W}$ Invalid	5	---	3	---	ns
18	Clock High to $R/\overline{W}$ High	0	15	0	15	ns
20	Clock High to $R/\overline{W}$ Low	0	15	0	15	ns
21 <sup>6</sup>	$R/\overline{W}$ High to $\overline{AS}$ Asserted	5	---	5	---	ns
22 <sup>6</sup>	$R/\overline{W}$ Low to $\overline{DS}$ Asserted (Write)	35	---	35	---	ns
23	Clock High to Data Out Valid	---	14	---	19	ns
24	Data Out Valid to Negating Edge of $\overline{AS}$	3	---	3	---	ns
25 <sup>6,11</sup>	$\overline{AS}$ , $\overline{DS}$ Negated to Data Out Invalid	5	---	5	---	ns
25A <sup>9,11</sup>	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	5	---	5	---	ns
26 <sup>6,11</sup>	Data Out Valid to $\overline{DS}$ Asserted (Write)	5	---	5	---	ns
27	Data-In Valid to Clock Low (Synchronous Setup)	1	---	6	---	ns
27A	Late $\overline{BERR}$ , $\overline{HALT}$ Asserted to Clock Low (Setup)	3	---	10	---	ns

#### C-4 Timing Comparisons

### 13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(V<sub>cc</sub> = 5.0 Vdc +/- 5%; GND = 0 Vdc; TA = 0° to 70° C)

Num	Characteristic	33.33 MHz <sup>15</sup>		HP 64430		Unit
		Min	Max	Min	Max	
28 <sup>12</sup>	$\overline{\text{AS}}$ , $\overline{\text{DS}}$ Negated to $\overline{\text{DSACKx}}$ , $\overline{\text{BERR}}$ , $\overline{\text{HALT}}$ , $\overline{\text{AVEC}}$ Negated (Asynchronous Hold)	0	30	0	18	ns
28A <sup>12</sup>	Clock Low to $\overline{\text{DSACKx}}$ , $\overline{\text{BERR}}$ , $\overline{\text{HALT}}$ , $\overline{\text{AVEC}}$ Negated (Synchronous Hold)	6	50	6	43	ns
29 <sup>12</sup>	$\overline{\text{DS}}$ Negated to Data-In Invalid (Asynchronous Hold)	0	---	0	---	ns
29A <sup>12</sup>	$\overline{\text{DS}}$ Negated to Data-In High Impedance	---	30	---	27	ns
30 <sup>12</sup>	Clock Low to Data-In Invalid (Synchronous Hold)	6	---	6	---	ns
30A <sup>12</sup>	Clock Low to Data-In High Impedance (Read followed by Write)	---	45	---	35	ns
31 <sup>2</sup>	$\overline{\text{DSACKx}}$ Asserted to Data-In Valid (Asynchronous Data Setup)	---	20	---	20	ns
31A <sup>3</sup>	$\overline{\text{DSACKx}}$ Asserted to $\overline{\text{DSACKx}}$ Valid (Skew)	---	5	---	5	ns
32	$\overline{\text{RESET}}$ Input Transition Time	---	1.5	---	1.5	Clks
33	Clock Low to $\overline{\text{BG}}$ Asserted	0	15	0	24	ns
34	Clock Low to $\overline{\text{BG}}$ Negated	0	15	0	24	ns
35	$\overline{\text{BR}}$ Asserted to $\overline{\text{BG}}$ Asserted ( $\overline{\text{RMC}}$ Not Asserted)	1.5	3.5	1.5	3.5	Clks
37	$\overline{\text{BGACK}}$ Asserted to $\overline{\text{BG}}$ Negated	1.5	3.5	1.5	3.5	Clks
37A	$\overline{\text{BGACK}}$ Asserted to $\overline{\text{BR}}$ Negated	0	1.5	0	1.5	Clks

### 13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; TA = 0° to 70° C)

Num	Characteristic	33.33 MHz <sup>15</sup>		HP 64430		Unit
		Min	Max	Min	Max	
39 <sup>6</sup>	$\overline{\text{BG}}$ Width Negated	45	---	43	---	ns
39A	$\overline{\text{BG}}$ Width Asserted	45	---	43	---	ns
40	Clock High to $\overline{\text{DBEN}}$ Asserted (Read)	0	18	0	19	ns
41	Clock Low to $\overline{\text{DBEN}}$ Negated (Read)	0	18	0	19	ns
42	Clock Low to $\overline{\text{DBEN}}$ Asserted (Write)	0	18	0	19	ns
43	Clock High to $\overline{\text{DBEN}}$ Negated (Write)	0	18	0	19	ns
44	R/ $\overline{\text{W}}$ Low to $\overline{\text{DBEN}}$ Asserted (Write)	5	---	5	---	ns
45 <sup>5</sup>	$\overline{\text{DBEN}}$ Width Asserted (Asynchronous Read)	30	---	28	---	ns
	$\overline{\text{DBEN}}$ Width Asserted (Asynchronous Write)	60	---	58	---	ns
45 <sup>9</sup>	$\overline{\text{DBEN}}$ Width Asserted (Synchronous Read)	5	---	5	---	ns
	$\overline{\text{DBEN}}$ Width Asserted (Synchronous Write)	30	---	28	---	ns
46	R/ $\overline{\text{W}}$ Width Asserted (Asynchronous Write or Read)	75	---	75	---	ns
46A	R/ $\overline{\text{W}}$ Width Asserted (Synchronous Write or Read)	45	---	45	---	ns
47A	Asynchronous Input Setup Time ( $\overline{\text{HALT}}$ , $\overline{\text{BERR}}$ , $\overline{\text{DSACKx}}$ )	2	---	9	---	ns
	Asynchronous Input Setup Time ( $\overline{\text{IPLx}}$ )	2	---	12	---	ns
47B	Asynchronous Input Hold Time from Clock Low	6	---	6	---	ns
48 <sup>4</sup>	$\overline{\text{DSACKx}}$ Asserted to $\overline{\text{BERR}}$ , $\overline{\text{HALT}}$ Asserted	---	18	---	16	ns
53	Data Out Hold from Clock High	2	---	2	---	ns

#### C-6 Timing Comparisons



### 13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES (Cont'd)

(V<sub>cc</sub> = 5.0 Vdc +/- 5%; GND = 0 Vdc; TA = 0° to 70° C)

Num	Characteristic	33.33 MHz <sup>15</sup>		HP 64430		Unit
		Min	Max	Min	Max	
55	R/W Asserted to Data Bus Impedance Change	15	---	12	---	ns
56	RESET Pulse Width (Reset Instruction)	512	---	512	---	Clks
57	BERR Negated to HALT Negated (Rerun)	0	---	2	---	ns
58 <sup>10</sup>	BGACK Negated to Bus Driven	1	---	1	---	Clks
59 <sup>10</sup>	BG Negated to Bus Driven	1	---	1	---	Clks
60 <sup>13</sup>	Synchronous Input Valid to Clock High (Setup Time)	2	---	4	---	ns
61 <sup>13</sup>	Clock High to Synchronous Input Invalid (Hold Time)	6	---	6	---	ns
62	Clock Low to STATUS, REFILL Asserted	0	15	0	25	ns
63	Clock Low to STATUS, REFILL Negated	0	15	0	25	ns

MC68030 electrical specifications reprinted courtesy Motorola, Inc.

#### NOTES:

1. This number can be reduced to 5 nanoseconds if strobes have equal loads.
2. If the asynchronous setup time (#47A) requirements are satisfied, the DSACKx low to data setup time (#31) and DSACKx low to BERR low setup time (#48) can be ignored. The data must only satisfy the data-in to clock low setup time (#27) for the following clock cycle and BERR must only satisfy the late BERR low to clock low setup time (#27A) for the following clock cycle.
3. This parameter specifies the maximum allowable skew between DSACK0 to DSACK1 asserted or DSACK1 to DSACK0 asserted; specification #47A must be met by DSACK0 or DSACK1.

4. This specification applies to the first  $\overline{\text{DSACKx}}$  signal asserted. In the absence of  $\overline{\text{DSACKx}}$ ,  $\overline{\text{BERR}}$  is an asynchronous input using the asynchronous input setup time (#47A).
5.  $\overline{\text{DBEN}}$  may stay asserted on consecutive write cycles.
6. The minimum values must be met to guarantee proper operation. If this maximum value is exceeded,  $\overline{\text{BG}}$  may be reasserted.
7. This specification indicates the minimum high time for  $\overline{\text{ECS}}$  and  $\overline{\text{OCS}}$  in the event of an internal cache hit followed immediately by a cache miss or operand cycle.
8. This specification guarantees operation with the MC68881/MC68882, which specifies a minimum time for  $\overline{\text{DS}}$  negated to  $\overline{\text{AS}}$  asserted (specification #13A in the Motorola MC68881/MC68882 User's Manual). Without this specification, incorrect interpretation of specifications #9A and #15 would indicate that the MC68030 does not meet the MC68881/MC68882 requirements.
9. This specification allows a system designer to guarantee data hold times on the output side of data buffers that have output enable signals generated with  $\overline{\text{DBEN}}$ . The timing on  $\overline{\text{DBEN}}$  precludes its use for synchronous read cycles with no wait states.
10. These specifications allow system designers to guarantee that an alternate bus master has stopped driving the bus when the MC68030 regains control of the bus after an arbitration sequence.
11.  $\overline{\text{DS}}$  will not be asserted for synchronous write cycles with no wait states.
12. These hold times are specified with respect to strobes (asynchronous) and with respect to the clock (synchronous). The designer is free to use either hold time.
13. Synchronous inputs must meet specifications #60 and #61 with stable logic levels for all rising edges of the clock. These values are specified relative to the high level of the rising clock edge.
14. This specification allows system designers to qualify the  $\overline{\text{CS}}$  signal of an MC68881/MC68882 with  $\overline{\text{AS}}$  (allowing 7ns for a gate delay) and still meet the  $\overline{\text{CS}}$  to  $\overline{\text{DS}}$  setup time requirement (specification 8B) of the MC68881/MC68882.
15. The clock signal used during test has 5ns of rise time and 5ns of fall time. For system implementations that have less clock rise and fall times, the clock pulse width minimum should be commensurately longer so that: system  $(t_2 + (t_4 + t_5/2))$  is = or greater than minimum  $t_1/2$  and system  $(t_3 + (t_4 + t_5/2))$  is = or greater than minimum  $t_1/2$ .

## C-8 Timing Comparisons

## HP 64430 DC Electrical Specifications

(V<sub>CC</sub> = 5.0 Vdc +/- 5%; GND = 0 Vdc; T<sub>A</sub> = 0° to 70° C)

Characteristic	Symbol	Min	Max	Unit
Input High Voltage	V <sub>IH</sub>	2.0	V <sub>CC</sub>	V
Input Low Voltage	V <sub>IL</sub>	-0.5	0.8	V
Input Leakage Current GND = or < V <sub>in</sub> = or < V <sub>CC</sub>	<u>BR</u> , <u>BGACK</u> , <u>IPLx</u> , <u>MMUDIS</u> , <u>CDIS</u> , I <sub>in</sub>	-2.5	2.5	uA
Input High Current	<u>CBACK</u> , <u>CIIN</u> , <u>STERM</u> I <sub>IH</sub> <u>BERR</u> , <u>AVEC</u> , <u>DSACKx</u> , <u>HALT</u> CLK, RESET	---	0 25 50	uA
Input Low Current	<u>RESET</u> , <u>CBACK</u> , <u>CIIN</u> , <u>STERM</u> I <sub>IL</sub> CLK, <u>BERR</u> , <u>AVEC</u> , <u>DSACKx</u> , <u>HALT</u>	---	-1.4 -0.25	mA
Output High Voltage I <sub>OH</sub> = -400uA	A0-A31, <u>AS</u> , <u>BG</u> , <u>D0-D31</u> , <u>DBEN</u> , <u>DS</u> , <u>ECS</u> , V <sub>OH</sub> <u>R/W</u> , <u>STATUS</u> , <u>REFILL</u> , <u>IPEND</u> , <u>OCS</u> , <u>RMC</u> , <u>SIZ0-SIZ1</u> , <u>FC0-FC2</u> , <u>CBREQ</u> , <u>CIOUT</u>	2.4	---	V
Output Low Voltage I <sub>OL</sub> = 2.5mA I <sub>OL</sub> = 3.2 mA I <sub>OL</sub> = 4.5mA I <sub>OL</sub> = 5.3 mA I <sub>OL</sub> = 2.0 mA I <sub>OL</sub> = 9.3 mA	A0-A31, <u>FC0-FC2</u> , <u>SIZ0-SIZ1</u> V <sub>OL</sub> <u>BG</u> , <u>D0-D31</u> <u>CBREQ</u> , <u>R/W</u> , <u>RMC</u> <u>AS</u> , <u>DS</u> , <u>DBEN</u> , <u>IPEND</u> <u>STATUS</u> , <u>REFILL</u> , <u>CIOUT</u> , <u>ECS</u> , <u>OCS</u> RESET	---	0.5 0.5 0.5 0.5 0.5 0.5	V
Power Dissipation	T <sub>A</sub> = 0° C P <sub>D</sub> T <sub>A</sub> = 70° C	---	0 0	W
Capacitance (V <sub>IN</sub> = 0V, T <sub>A</sub> = 25° C, f = 1MHz)	C <sub>in</sub>	---	20	pF
Load Capacitance	C <sub>L</sub>	---	100 50	pF

---

# Notes

# Index

---

- I
  - .EA configuration file ,2-13
  - .EA file ,6-29
  - .EB configuration files ,2-13
- A
  - absolute files, loading ,6-27
  - accessing the emulation system ,3-9
  - address range overlays ,4-29
  - address specification, custom register ,8-4
  - address translation cache (ATC), flushing ,5-3
  - analysis with cache enabled ,6-15
  - analyzer ,6-29 - 6-30
  - analyzer, debugging plug-in failures ,6-30
  - analyzer, use for debugging ,6-30
  - are you there function, how it works ,10-3
- B
  - background ,1-5
  - background monitor ,7-3
  - background monitor, defaulting stack pointer for ,4-14
  - BERR, enabling/disabling ,6-19
  - block size ,4-28
  - blocking target BERR during emulation memory cycles ,4-25
  - break function, breaking into the monitor ,7-6
  - break on write to ROM ,1-3
  - breakpoints ,1-4
  - bus cycle ,6-30
  - bus cycle, none for more than 250 milliseconds ,6-30
  - bus size bit (B) ,6-30
- C
  - cache control ,6-14
  - caches, using the ,6-14
  - card cage access cover, removing the ,2-6
  - clock rates, CPU ,4-47
  - code branches, incorrect ,6-30
  - command modules, emulation monitor ,7-10
  - command scanner ,7-9
  - comparison of foreground/background monitors ,7-3
  - compiling the demonstration programs ,3-8

- configuration file name ,4-51
- configuration file, .EA ,2-13
- configuration file, .EB ,2-13
- configuration file, starting with the default ,6-29
- configuration file, viewing ,6-29
- configuration file, what to do after modifying ,2-13
- configuration file, incorrect ,6-29
- configuration, deMMUer ,4-41
- configuration, review ,6-29
- configuring simulated I/O ,9-2
- connecting the emulator pod to your target system ,2-10
- continuing target system interrupts
  - while in the emulation monitor ,7-18
- controlling flow of data and code ,4-26
- coprocessor configuration questions ,8-13
- coprocessor copy routine ,8-11
- coprocessor register buffer, emulation monitor ,8-9
- copying from target memory, how it works ,10-16
- copying the demonstration programs to your subdirectory ,3-7
- copying to target system memory, how it works ,10-17
- CPU clock rates faster than 25 MHz ,4-47
- cpu registers, modifying, how it works ,10-19
- custom coprocessor, register set display specification ,8-5
- custom coprocessors support ,1-5
- custom coprocessors, modifying configuration for ,4-20
- custom register address specification ,8-4
- custom register format file ,8-3
- custom register format file, specifying ,4-22
- custom register name specification ,8-4
- custom register size specification ,8-4
- custom registers, emulation monitor changes ,8-9
- customizing the emulation monitor ,7-12

**D**

- debugging plug-in failures with analyzer ,6-30
- debugging plug-in problems ,6-29
- debugging, use status messages ,6-30
- default response to emulation configuration questions ,4-3
- defaulting stack pointer for background monitor ,4-14
- deleting memory map entries ,4-41

- deMMUer ,1-4
  - addresses for which it has no translation ,5-3
  - how it operates ,5-2
  - how to access the deMMUer configuration display ,5-10
  - how to turn off ,5-7
  - how to turn on ,5-7
  - startup with the emulator ,5-6
  - turn on/off by setting the analysis mode ,5-8
  - turn on/off using emulation configuration questions ,5-7
  - what it is ,5-2
  - when it will not do reverse-address translations ,5-5
  - when to start ,5-6
  - when to turn off ,5-5
  - when to use ,5-4
- deMMUer configuration ,4-41
- displaying cpu registers, how it works ,10-18
- displaying global symbols ,3-12
- displaying local symbols, example ,3-13
- displaying memory ,3-14
- displaying registers ,3-17
- displaying target memory, how it works ,10-15
- dividing the processor address space ,4-26
- DMA enable/disable ,6-19
- dma transfers into emulation memory ,4-46
- dma transfers, enabling ,4-45
- DSACK and STERM, interlocking
  - emulation memory and target ,6-10
- dsack DSACK signal problems, open
  - collector drivers on DSACK line ,6-11
- DSACK signal problems, early removal of DSACK signals ,6-12
- DSACK signal problems, isolating the problem ,6-12
- DSACK signal problems, target system ,6-11
- DSACK signals, using ,6-10

**E**

- emulation configuration, modifying the default ,3-9
- emulation features
  - custom coprocessors support ,1-5
  - foreground or background mmonitor ,1-5
  - function codes ,1-5
  - memory management ,1-4
  - out-of-circuit or in-circuit emulation ,1-6
- emulation memory ,4-29

- emulation memory breakpoints with cache enabled ,6-16
- emulation memory display operations ,4-29
- emulation memory load operations ,4-29
- emulation memory, loading ,3-11
- emulation monitor
  - foreground or background ,1-5
  - monitor ,1-5
- emulation monitor changes for custom coprocessors ,8-9
- emulation monitor description ,7-7
- emulation monitor entry point routines ,7-8
- emulation monitor functions, enabling ,4-6
- emulation monitor memory requirements ,4-29
- emulation monitor, coprocessor register buffer ,8-9
- emulation monitor, loading ,6-24, 7-22
- emulation pod configuration, modifying ,4-43
- emulation system components, example system ,3-2
- emulation system hardware, installing ,2-5
- emulation system, accessing ,3-9
- emulation system, preparing ,3-8
- emulator
  - purpose ,1-2
- emulator features
  - breakpoints ,1-4
  - processor reset control ,1-4
  - register display/modify ,1-4
  - restrict to real-time runs ,1-2
  - single-step processor ,1-4
- emulator pod cables, connecting to the emulator boards ,2-7
- emulator pod, connecting to the target system ,2-10
- emulator use of int7, enabling ,4-9
- emulator use of software breakpoints, enabling ,4-12
- enabling the foreground monitor ,4-18
- ending the emulation session ,3-34
- ending the mapping session ,4-42
- entering mapper blocks ,4-30
- entering mapper blocks, syntax ,4-30
- entry point routines, emulation monitor ,7-8
- error messages ,A-1
- examples, emulation system used for ,3-2
- exception vector table ,7-7
- EXCEPTION\_ENTRY emulation monitor routine ,7-9



- executing a software breakpoint, how it works ,10-8
- external hardware features of the instrumentation cardcage ,2-1
- F**
  - failures, plug-in ,6-29
  - flush the address translation cache ,5-3
  - foreground ,1-5
  - foreground monitor ,7-3
  - foreground monitor, enabling ,4-18
  - foreground monitor, interlock or provide termination for ,4-19
  - function codes ,1-5
- G**
  - getting started, linking modules ,3-6
  - guarded memory access ,4-4
- H**
  - halt ,6-30
  - hardware installation instructions ,2-5
  - how does a simulated interrupt function ,9-5
- I**
  - i/o operations ,4-29
  - illegal conditions ,4-4
  - initializing and configuring your measurement system ,3-4
  - inspecting the equipment ,2-4
  - installing boards into the card cage ,2-8
  - installing emulation system hardware ,2-5
  - installing hardware, instructions ,2-5
  - installing software ,2-13
  - installing software updates ,2-13
  - instructions on installing hardware ,2-5
  - interlock the foreground monitor ,4-19
  - interlocking emulation memory DSACK
    - and STERM and target DSACK and STERM ,6-10
  - ipend, enabling target line during emulator breaks ,4-11
  - isolate problems ,6-29
- J**
  - JSR\_ENTRY emulation monitor routine ,7-9
- L**
  - leading zeros ,4-32
  - linker listing file, example ,7-21
  - linking the emulation foreground monitor ,7-22
  - linking the program modules ,3-6
  - loading emulation memory ,3-11
  - loading the emulation monitor ,6-24, 7-22
  - logical (virtual) address to physical address translation ,5-2

- M**
  - making a subdirectory for your 68030 project ,3-2
  - mapper blocks, syntax for entering ,4-30
  - mapping display softkey labels ,4-28
  - mapping memory ,4-26
  - memory access timing issues ,6-26
  - memory default ,4-28
  - memory management ,1-4
  - memory map definition ,4-28
  - memory map display entries ,4-27
  - memory map example ,4-34
  - memory requirements, emulation monitor ,7-21
  - MMU table walks, deMMUer tracking ,5-2
  - modify default memory ,4-40
  - modify defined\_codes ,4-37
  - Modify memory configuration? ,4-24, 4-43
  - modifying a memory configuration ,4-23
  - modifying memory ,3-15
  - modifying target memory, how it works ,10-16
  - modifying the configuration file ,4-2
  - modifying the cpu registers, how it works ,10-19
  - modifying the default emulation configuration ,3-9
  - modifying the emulation monitor exception vector table ,7-14
  - modifying the emulation monitor to use simulated interrupts ,9-11
  - modifying the memory map ,4-37
  - modifying the MON\_ALT\_BUFFER table ,8-10
  - modifying the MON\_ALT\_REGISTERS table ,8-11
  - monitor (emulation)
    - background ,7-3
    - comparison of foreground/background ,7-3
    - foreground ,7-3
  - monitor message routine, example ,7-19
  - MONITOR\_ENTRY emulation monitor routine ,7-8
  - MONITOR\_ENTRY, foreground monitor label ,1-4
  - msinit ,2-13
- N**
  - name specification, custom register ,8-4
  - naming the configuration file ,4-51
  - NMI ,7-6
- O**
  - on-chip cache disabling ,4-48
  - operational overview ,3-1

- P**
  - partitioning the processor address space ,4-28
  - performance ,6-31
  - physical address to logical address translation ,5-2
  - plug-in failures ,6-29
  - plug-in problems, debugging ,6-29
  - pod cable, securing ,2-10
  - preinstallation inspection ,2-4
  - preparing the emulation system ,3-8
  - preparing your program modules, getting started ,3-6
  - problems, isolate ,6-29
  - purpose of the emulator ,1-2
  
- R**
  - real-time runs ,1-2
  - real-time/nonreal-time run mode, selecting ,4-4
  - register display/modify ,1-4
  - register set display specification, custom coprocessor ,8-5
  - removing the development environment card cage access cover ,2-6
  - reserved address space, using function codes with ,6-17
  - reset control ,1-4
  - RESET\_ENTRY emulation monitor routine ,7-9
  - resetting into the monitor ,4-8, 6-24
  - restoring the processor interrupt mask ,7-18
  - restrict to real-time runs ,1-2
  - restrictions on using simulated I/O ,9-4
  - restrictions on using simulated interrupts ,9-10
  - run command after a software breakpoint, how it works ,10-8
  - run command, how it works ,10-4
  - run from ... until command, how it works ,10-5
  - run from ... until command, using ,6-22
  - run from command, how it works ,10-4
  - run until command, how it works ,10-5
  - running emulation ,4-2
  - running from the transfer address ,3-16
  
- S**
  - safety considerations ,1-1, 2-3
  - sample emulation configuration command file ,3-16
  - securing the pod cable ,2-10
  - sending messages from the user
    - program to the emulator display ,7-18
  - shutdown ,6-30
  - simint.c source file ,B-9
  - simulated I/O, configuring ,9-1 - 9-2

- simulated I/O, restrictions on using ,9-4
- simulated interrupt, how they function ,9-5
- simulated interrupts ,9-5
- simulated interrupts, modifying the monitor to use ,9-11
- simulated interrupts, restrictions on using ,9-10
- single stepping with background monitor, how it works ,10-11
- single stepping with foreground monitor, how it works ,10-10
- single-step processor ,1-4
- size specification, custom register ,8-4
- software breakpoint instruction number selection ,4-12
- software breakpoint, setting ,10-7
- software breakpoints ,1-4, 10-7
- software breakpoints, using ,3-24
- stack pointer for background monitor, defaulting ,4-14
- starting address of a block boundary ,4-32
- status conditions, incorrect ,6-30
- status messages, use for debugging ,6-30
- step function, using ,3-19
- STERM signals, using ,6-10
- SWBK\_ENTRY emulation monitor routine ,7-8
- symbolic debugging ,1-2
- symbols ,1-2

**T**

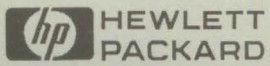
- table search ,5-3
- target memory ,4-29
- target memory breakpoints with cache enabled ,6-16
- target memory display operations ,4-29
- target memory load operations ,4-29
- target memory transfers, how it works ,10-12
- target memory, copying from, how it works ,10-16
- target memory, modifying, how it works ,10-16
- target system ,1-2
- target system interface ,6-1
- target system memory, copying to, how it works ,10-17
- target system program interrupt ,7-6
- target system, connecting to the emulator pod ,2-10
- terminate the foreground monitor ,4-19
- timing issues, memory access ,6-26
- towers.c source file ,B-2
- tracing processor activity ,3-20
- transfer address, running from ,3-16
- translation tables ,5-3

translation, logical (virtual) address to physical address ,**5-2**  
translation, physical address to logical address ,**5-2**  
trigger pulse ,**6-30**

- U** unpacking the equipment ,**2-4**
  - using breakpoints with caches enabled ,**6-15**
  - using command files ,**3-34**
  - using simulated i/o, example ,**3-27**
  - using the emulation monitor ,**6-24**
  - using the emulator ,**3-11**
  - using the modify memory map command ,**4-37**
- V** vector base register,use of ,**6-13**
- W** writing coprocessor copy routines ,**8-11**

---

# Notes



HEWLETT  
PACKARD

Printed In U.S.A.