HEWLETT PACKARD

User's Guide

HP 64430 68030 Emulation

HP 64430

68030 Emulator

User's Guide

HEWLETT PACKARD

HP Part No. 64430-97008
Printed in U.S.A.
June 1991

Edition 3

Certification and Warranty

Certification

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

Warranty

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

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

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

Limitation of Warranty

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

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

Exclusive Remedies

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

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

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

Notice

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

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

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

Printing History

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

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

Edition 1	64430-97000, February 1990
Edition 2	64430-97007, February 1991
Edition 3	**64430-97008, June 1991**

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 ferrules, or brackets that ground the cable shields are clean and tight (do not over-tighten). The 68030 Emulator cables have exposed shields that must make contact with the cable clamp.

3. During times of infrequent use, disconnect the 68030 Emulator 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. Westerlin betrieben wird, senden Sie bitte die beiliegende Postkarte ausgefüllt an Ihr zuständiges Fernmeldeamt.

Notes

Safety

Summary of Safe Procedures

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

Ground The Instrument

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

Do Not Operate In An Explosive Atmosphere

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

Keep Away From Live Circuits

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

Designed to Meet Requirements of IEC Publication 348

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

Do Not Service Or Adjust Alone

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

Do Not Substitute Parts Or Modify Instrument

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

Dangerous Procedure Warnings

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

Warning

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

Safety Symbols Used In Manuals

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

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

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

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

 OR 

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.

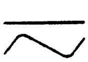
Low-noise or noiseless, clean ground (earth) terminal. Used for a signal common, as well as providing protection against electrical shock in case of a fault. A terminal marked with this symbol must be connected to ground in the manner described in the installation (operating) manual before operating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

 OR 

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 during shipment with either a conductive foam wafer or a conductive plastic pin protector. This is done for two reasons:

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

Because the protection devices are conductive, the emulator may not function correctly during normal operation or option_test performance verification. You should remove the foam or plastic device before using the emulation or analysis system or before running option_test performance verification.

When you're not using the emulator, replace the foam or plastic assembly to retain protection for the probe pins and protection from ESD.

Notes

Using this Manual

Organization

Chapter 1 "Introducing The 68030 Emulator" contains a brief description of the 68030 emulator.

Chapter 2 "Installing Emulation Hardware" tells how to install the 68030 emulation system hardware into the instrumentation cardcage. It explains how to make a measurement system. This chapter also tells how to connect 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.

Chapter 4 "Configuring Your Emulator" shows how to:

- Access the emulation configuration questions.
- Load configuration files from a previous emulation session.

It also describes each configuration option in detail.

Chapter 5 "DeMMUer - What It Is And How It Works" describes:

- What the deMMUer is.
- How the deMMUer operates.
- When to use the deMMUer.
- Restrictions you need to observe when using the deMMUer.

Chapter 6 "Target System Interface" describes the 68030 signals and how the emulator interacts with those signals. It also gives guidelines for using the emulator with a target system and tells you how the emulator interacts with your target system.

Chapter 7 "The Emulation Monitor Program" describes the emulation monitor program and tells 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 simulate target system I/O and how to use the simulated interrupt features.

Chapter 10 "How The Emulator Works" describes the implementation of many emulator features. Understanding the emulator helps you use it more effectively and can help you solve problems.

Appendix A "Emulation Error Messages" describes most error messages you might encounter and tells how to correct the errors.

Appendix B "Timing Comparisons" lists timing comparisons between 68030 processors and the HP 64430 Emulator. It also gives 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 started with 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 in this manual use the following structure:

copy display to `trcfile1`

copy display to	Softkeys appear in bold italic type in examples. Commands appear in **bold** in text. You will not be prompted to use the **---ETC---** softkey to search for the appropriate softkey template. Three softkey templates are available at the HP 64000-UX system monitor level.
trcfile1	This is the name of a file, which you must type in. There are no softkeys for this type of selection since it is variable. However, a softkey prompt such as **<FILE>** will appear as a softkey selection.

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

Notes

Contents

1 **Introducing The 68030 Emulator**

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

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 2-9
Reinstall Card Cage Access Cover 2-9
Installing the Emulator Probe In the Target System 2-9
Install Software 2-12
Installing 68030 Emulation Software Updates 2-12
Turning On the HP 64120A 2-13

3 **Getting Started**

Overview 3-1
Introduction 3-1
Emulation System Used For Examples 3-1
Make A Subdirectory For Your 68030 Project 3-2
Initialize And Configure Your Measurement System 3-4
Prepare Your Program Modules 3-6
Create The Absolute File In Your Subdirectory 3-8
Use The Absolute File In The Demo Directory 3-8
Prepare The Emulation System 3-9
Access The Emulation System 3-9
Modify The Default Emulation Configuration 3-10
Load Emulation Memory 3-11
Use The Emulator 3-12
Display The Source File 3-12
Symbol Handling 3-13
Displaying Global Symbols 3-14
Displaying Local Symbols 3-15
Display Memory 3-19
Adding Symbols To The Memory Display 3-20
Adding Source-File Lines To The Memory Display 3-21
Modify Memory 3-22
Run From The Transfer Address 3-23
Display Registers 3-24
Use The Step Function 3-26
Stepping Through The Program In Memory 3-27
Trace Processor Activity 3-29
Use Software Breakpoints 3-32
Using Simulated I/O 3-35
Ending The Emulation Session 3-37
Using Command Files 3-37
Use The DeMMUer 3-38
End Of DeMMUer Demonstration 3-40

4 Answering Emulation Configuration Questions

Overview 4-1
Introduction 4-1
Running Emulation 4-2
Modify the Configuration File 4-2
- Selecting Real-Time/ Nonreal-Time Run Mode 4-3
- Enabling Emulator Monitor Functions 4-4
- Reset Into the Monitor 4-5
- Enabling Emulator Use of Software Breakpoints 4-7
- Selecting the Software Breakpoint Instruction Number 4-7
- Defaulting the Stack Pointer For the Background Monitor . 4-8
- Select To Block ECS, OCS Signals During Background Monitor Cycles 4-8
- Choose To Perform Periodic Foreground Accesses 4-8
- Selecting Address for Periodic Foreground Access 4-9
- Enabling the Foreground Monitor 4-9
- Interlock or Provide Termination for the Foreground Monitor 4-9
- Using Custom Coprocessors 4-10
- Specifying The Custom Coprocessor File 4-10
- Modifying a Memory Configuration 4-11
- Break on Write to ROM 4-12
- Selecting to Block BERR on Non-interlocked Emulation Memory 4-12
  - Mapping Memory 4-13
    - Memory Map Display Organization. 4-14
    - Memory Map Definition. 4-15
    - Emulation Monitor Program Memory Requirements. 4-16
  - Using The Map Command 4-16
  - Using the map_overlay Command 4-19
  - Memory Mapping Example 4-21
  - Using the modify Command 4-23
    - Modify Defined_Codes. 4-23
    - Modify <ENTRY>. 4-25
    - Modify Default. 4-26
  - Deleting Memory Map Entries 4-27
- Modify the DeMMUer Configuration 4-27
  - Ending The Mapping Session 4-28
- Modifying The Emulation Pod Configuration 4-28
- Configuring for In-circuit Emulation Session 4-29

Enabling DMA Transfers 4-29
Enabling DMA Transfers Into Emulation Memory 4-30
CPU Clock Rate Determination of Wait States 4-30
Disabling On-chip Cache 4-31
Enabling MMU For Use During Emulation Session 4-31
Modifying Simulated I/O Configuration 4-32
Modifying Simulated Interrupt Configuration 4-32
Naming The Configuration File 4-32

5 DeMMUer - What It Is And How It Works

Overview 5-1
Introduction 5-1
What The DeMMUer Is 5-2
How The DeMMUer Operates 5-2
When To Use The DeMMUer 5-3
When To Turn Off The DeMMUer 5-4
Unable to Do Reverse-Address Translations 5-4
When To Start The DeMMUer 5-5
Startup With The Emulator 5-5
Used Emulator without DeMMUer, Want To Use It Now . . 5-5
How To Turn On And Turn Off The DeMMUer 5-5
Turn On/Off By Using Configuration Questions 5-6
Turn On/Off By Setting The Analysis Mode 5-6
DeMMUer Configuration Setup 5-7
How To Access The DeMMUer Configuration Display 5-8

6 Target System Interface

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

DSACK1-DSACK0 6-5
BERR 6-5
HALT, AVEC 6-5
STERM 6-5
CIIN 6-6
CBACK 6-6
BG 6-6
IPEND 6-6
STATUS, REFILL 6-7
BR, BGACK 6-7
IPL2-IPL0 6-7
CDIS, MMUDIS 6-7
RESET 6-8
VCC 6-8
Emulation And Target System DSACK and STERM Signals . .6-8
Interlocking Emulation Memory and Target DSACK and STERM Signals 6-8
DSACK and STERM Signal Problems In Target Systems . 6-10
Use Of Open Collector Drivers 6-10
Early Removal Of DSACK Signals 6-10
Isolating The DSACK Problem 6-11
Using the Vector Base Register 6-11
Using the Internal 68030 Caches 6-12
Cache Control 6-12
Analysis with Cache 6-13
Using Breakpoints With Caches Enabled 6-13
Target Memory Breakpoints 6-14
Emulation Memory Breakpoints 6-14
Function Codes For Reserved Address Space 6-15
Enabling/Disabling BERR 6-16
Using DMA 6-16
Using the Run From ... Until Command 6-19
Using the Foreground Monitor 6-21
Loading the Monitor 6-21
Resetting Into the Monitor 6-21
Memory Access Timing Issues 6-23
33 MHz 68030 Microprocessor 6-23
HP 64430 68030 Emulation System 6-23
Loading An Absolute File 6-24
Debugging Plug-in Problems 6-25
Review the Configuration 6-25

Use the Internal Analyzer 6-26
Use the Status Messages 6-27
Run Performance Verification (PV) 6-27
If All Else Fails 6-27

7 The Emulation Monitor Programs

Overview 7-1
Introduction 7-1
Comparison of Foreground and Background Monitors 7-2
Background Monitors 7-2
Foreground Monitors 7-3
Choose a Foreground or Background Monitor 7-3
When to Use the Background Monitor 7-3
When to Use the Foreground Monitor 7-4
Customizing the Monitor Programs 7-5
The Break Function and the Emulation Monitor 7-5
Emulation Monitor Description 7-5
The Exception Vector Table 7-6
Emulation Monitor Entry Point Routines 7-6
MONITOR_ENTRY 7-7
SWBK_ENTRY 7-7
JSR_ENTRY 7-7
RESET_ENTRY 7-7
EXCEPTION_ENTRY 7-7
Emulation Command Scanner 7-8
Emulation Command Execution Modules 7-8
ARE_YOU_THERE 7-8
EXIT_MONITOR 7-8
SYNCH_START_ENABLE 7-8
COPY_MEMORY 7-9
COPY_ALT_REG 7-9
MON_ALT_REGISTERS 7-9
SIMINT_ENABLE 7-9
SIMINT_DISABLE 7-9
SIM_INTERRUPT 7-10
Using and Modifying the Foreground Monitor 7-10
Modifying The Exception Vector Table 7-11
Continuing Target System Interrupts While in the Emulation Monitor 7-15
Sending User Program Messages to the Display 7-16
Monitor Memory Requirements 7-18

Linking the Emulation Foreground Monitor 7-19
Loading the Emulation Monitor 7-19
Using Reset Into Foreground Monitor 7-19

8 Using Custom Coprocessors

Overview 8-1
Introduction 8-1
The Custom Register Format File 8-2
  Address Specification 8-3
  Size Specification 8-3
  Name Specification 8-4
  Register Set Display Specification 8-4
Emulation Monitor Changes 8-7
  Defining a Coprocessor Register Buffer 8-7
  Modifying the MON_CPU_REGISTERS Table 8-8
  Modifying The MON_ALT_REGISTERS Table 8-8
  Writing Coprocessor Copy Routines 8-9
Answering Emulation Coprocessor Configuration
 Questions 8-10

9 Using Simulated I/O And Simulated Interrupts

Overview 9-1
Configuring Simulated I/O 9-1
  Restrictions On Simulated I/O 9-4
Simulated Interrupts 9-4
  How Does a Simulated Interrupt Function? 9-5
  Simulated Interrupts Versus Real Interrupts 9-7
  Simulated Interrupt Configuration 9-7
  Restrictions On Simulated Interrupts 9-9
Modifying The Monitor To Use Simulated Interrupts 9-10

10 How The Emulator Works

Overview 10-1
Introduction 10-1
Are You There Function? 10-1
The Run Command 10-2
  Run From Command 10-3
  Run Until Command 10-4

Run From ... Until Command 10-4
Software Breakpoints 10-5
Setting A Software Breakpoint 10-6
Executing A Software Breakpoint 10-6
Executing A Run Command After Executing A Software Breakpoint 10-7
run 10-7
run from ADDR 10-8
Single Stepping With Foreground Monitor 10-8
Single Stepping With Background Monitor 10-10
Target Memory Transfers 10-11
Displaying Target Memory 10-13
Copying from Target System Memory 10-15
Modifying Target Memory 10-15
Copying to Target System Memory 10-16
Displaying the CPU Registers 10-16
Modifying the CPU Registers 10-17

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-4
running A-5
running in monitor A-5
slow dev at a= XXXX (YY) A-5
SRU Error Messages A-6

B Timing Comparisons

Introduction B-1

Index

1

Introducing The 68030 Emulator

Overview

This chapter gives the following information:

- Safety considerations for your emulator.
- Purpose of the 68030 emulator.
- Features of the 68030 emulator.
- Information in this manual.

Safety Considerations

The HP 64000-UX Microprocessor Development Environment, 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 was 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 replaces the 68030 microprocessor in your target system so you can control operation of the microprocessor in your application hardware (called the *target system*). The 68030 emulator acts like the 68030 microprocessor, but 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. You can debug software without a target system by using the HP 64430 Emulator's emulation memory (up to 2 Mbytes).

Symbols

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

Real-Time Operation

In real-time mode, your program runs 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
- display or modification of registers.

Clock Speed

You can use 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

During emulator configuration, you assign blocks of memory to physical address ranges. This is called *memory mapping*. If the MMU is enabled, you must know the system's physical memory arrangement.

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 assign emulation or target system memory in 256-byte blocks. Blocks can be defined as:

- Emulation; RAM or ROM, interlocked, synchronous, asynchronous with a data port width of 8-bits, 16-bits or 32-bits.
- Target; RAM or ROM, bus error blocked, cache disabled, burst mode blocked.
- Guarded access.

See the "Answering Emulation Configuration Questions" chapter for information on memory mapping.

The 68030 emulator will attempt to break to the emulation monitor on accessing guarded memory. You can configure the emulator to break to the emulation monitor on a write to ROM.

Analysis

The integrated emulation bus analyzer provides real-time analysis of bus cycle activity. You can define break conditions based on address and data bus cycle activity.

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 64404 and HP 64405 integrated analyzers are integrated, making it easy to make both emulation and analysis measurements.

Registers

You can display or modify the 68030 CPU register contents. For example, you can modify the program counter (PC) value to control where the emulator starts a program run. You also can display or modify the 68030 MMU register contents.

Single-Step

You can direct the emulation processor to execute a single instruction or several 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. See "Single Stepping with Foreground

Monitor" and "Single Stepping with Background Monitor" paragraphs in chapter 10 for 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 postmortem analysis of the program execution. You also can 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. You can select any one of the eight 68030 software breakpoint instructions to be used by the emulator.

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 support for custom coprocessors in the background monitor mode. It does 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 also can display and modify the custom coprocessor registers.

Function Codes

The HP 64430 emulator supports the 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 to qualify addresses specified in commands.

Foreground or Background Emulation Monitor

The 68030 emulator comes with both a foreground and a background monitor. This allows you to choose the monitor that best supports your development needs:

- 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, the monitor program reads 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 acts as if it were part of the target program.

The monitor program also can execute in *background*. In this mode, foreground operation is suspended so that the 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 emulator can be used in multiple emulation systems using other HP 64000-UX Microprocessor Development Environment emulators.

Manual Coverage

This manual tells you how to operate the HP 64430 emulator for the 68030 processor. The manual also gives 68030 emulator specific information. The *68030 Emulation Reference Manual* has more information about using 32-bit emulation, including detailed syntactic descriptions of the emulation commands. Detailed operating information for the HP 64404 and HP 64405 integrated analyzers is in the *Analysis Reference Manual for 32-Bit Microprocessors* and the *68030 Analysis Specifics* manual.

2

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, see the *HP 64000-UX Installation and Configuration Manual* for instructions on installing the HP 64120A Instrumentation Cardcage. Also, refer to the preinstallation instructions given in this section. Then install the emulation system as instructed in this chapter.

Figure 2-1 identifies some key features of the HP 64120A Instrumentation Cardcage. The labels used in this figure are used throughout this manual. Note the location of the power switch. For more information on the hardware configuration, see the *Installation and Configuration Manual.*

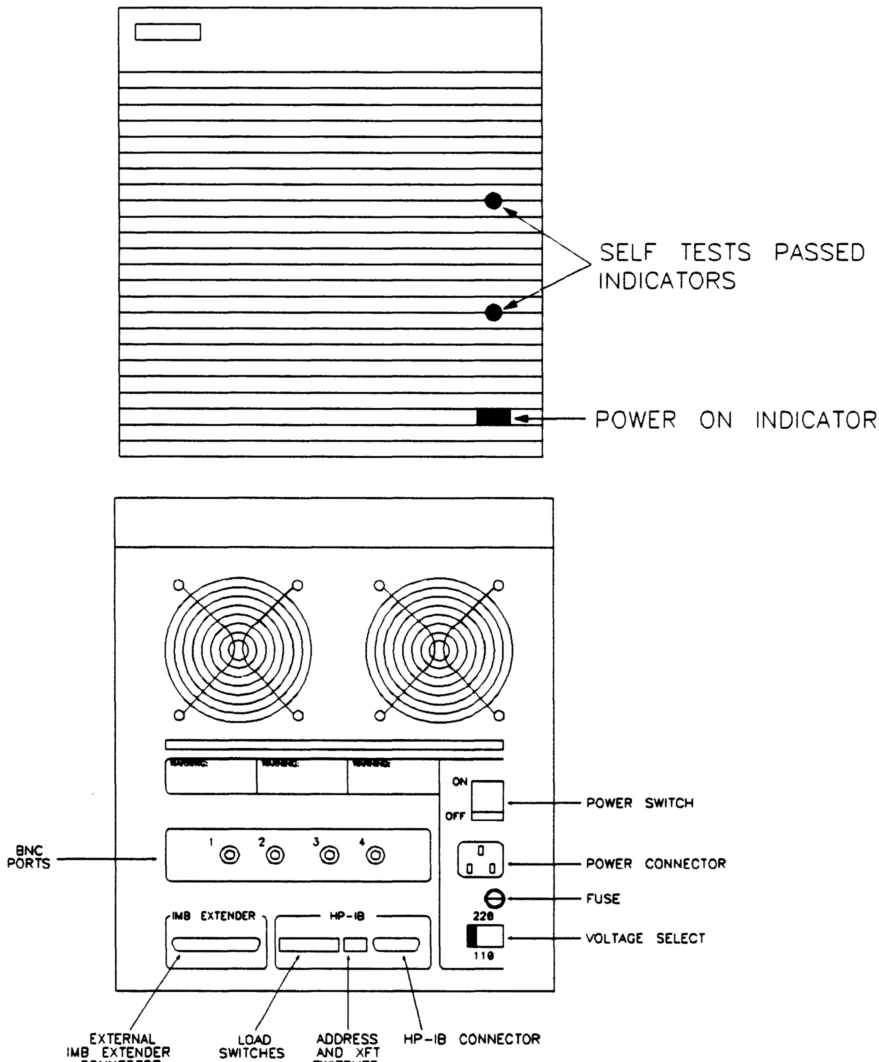


Figure 2-1. Instrumentation Cardcage Features

Safety Considerations

The HP 64000-UX Microprocessor Development Environment 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 was shipped in a safe condition.

You 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 done 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 emulation system circuit boards, cables, pod, and related equipment. Carefully inspect the equipment for shipping damage. If you find any damage, please contact your nearest Hewlett-Packard Sales/Service Office as soon as possible.

Make sure that you received everything that you ordered. 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

Follow these instructions 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 secured by four screws on the top of the instrumentation cardcage. See figure 2-2. Loosen the four screws, and remove the access cover.

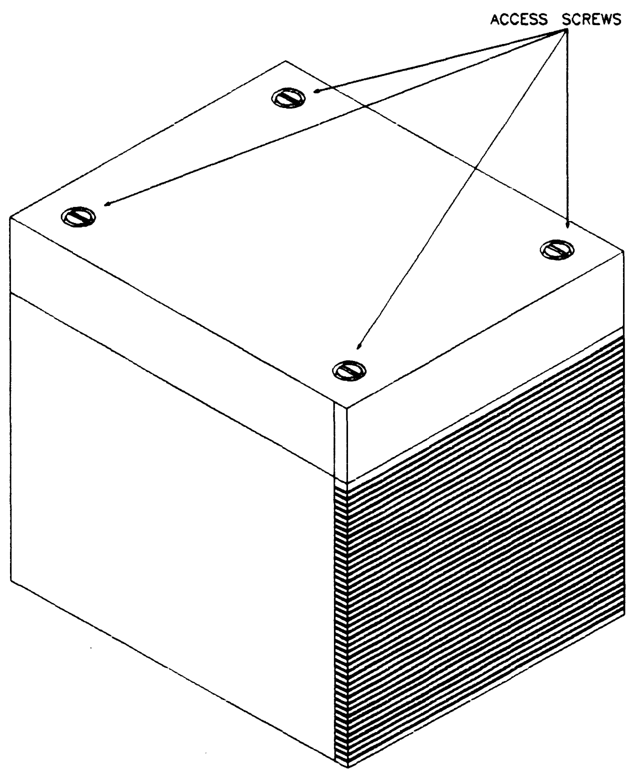


Figure 2-2. Removing the Cardcage Access Cover

Connect The Emulator Pod Cables To The Emulator Boards

There are six cables from the emulation pod that connect 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). 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 following the yellow, red, and brown color dots for proper connections.

   If you are using the DeMMUer board, connect the three 64-conductor cables from the pod to the DeMMUer board following the yellow, red, and brown color dots.

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, fasten the plastic cable cover to the board using four screws. See figure 2-3. Use a Torx TX 6 screwdriver.

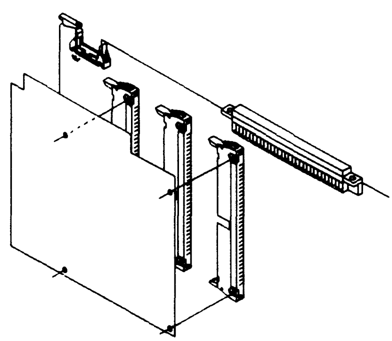


Figure 2-3. ABG Protective Plastic Cable Cover

Install Boards Into The Card Cage

Install the circuit boards 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 push down until the board seats in the motherboard connector. Be sure the ejector handles are horizontal when the board is seated.

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.

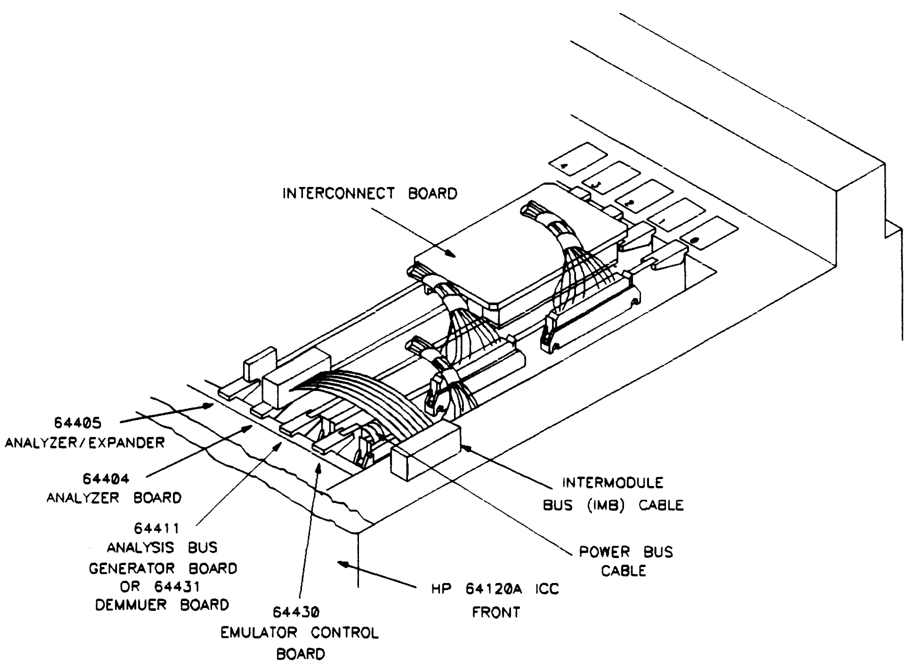


Figure 2-4. Board Installation Into Cardcage

The circuit boards need four adjacent card cage slots. 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. See 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.

Secure The Pod Cables

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

Reinstall Card Cage Access Cover

Reinstall the card cage access cover and fasten it with the hold-down screws.

Installing the Emulator Probe In the Target System

Caution

Possible damage to emulation probe! Protect against static discharge! The emulation probe contains devices that can be damaged by static discharge. Therefore, you should take precautions before handling the microprocessor connector to avoid damaging the internal probe components with 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. Otherwise, the pod may be damaged.

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. The emulator might be damaged 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. Also, turn off the target system first, then the development environment.

The emulation probe has 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. If the emulation probe is being installed on a densely loaded circuit board, there may not be enough room for 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.

1. Remove the 68030 processor from the target system processor PGA socket.

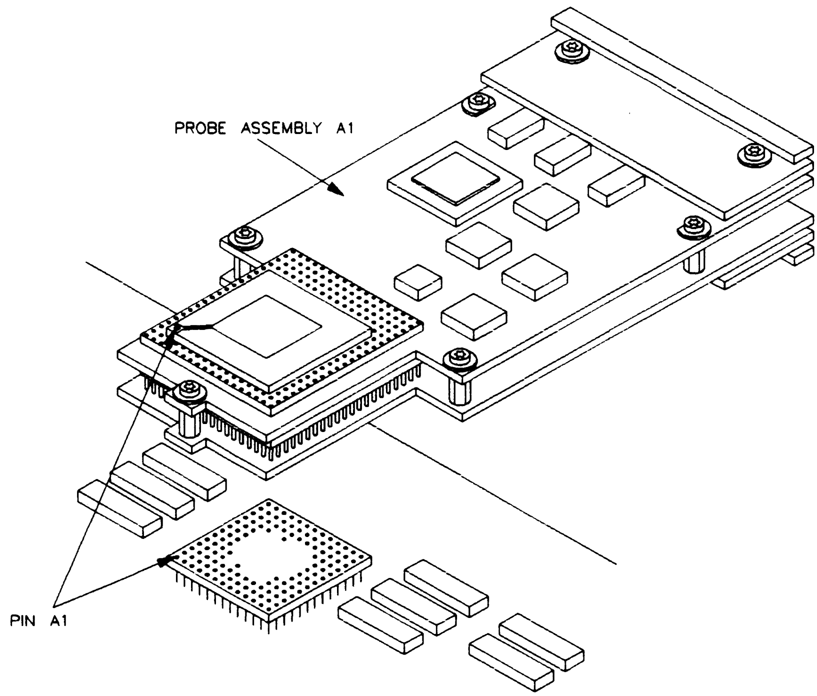


Figure 2-5. Installing Emulation Probe Into PGA Socket

2. Store the 68030 processor in a protected environment (such as antistatic foam). 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

See 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 begin your next emulation session. Refer to chapter 3 for a description of **msinit**.

When you install 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 a "modify configuration" session. You can edit this file 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. So 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.

Note

If you installed your HP 64430 emulation software as an update, remove the HP-OMF format absolute files from the demo directory with the command:

```
rm /usr/hp64000/demo/emul32/hp64430/*.X
```

Note that this is a capital X, not a small x.

Turning On the HP 64120A

Figure 2-1 shows the power switch for the HP 64120A Instrumentation Cardcage.

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. Also, 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 show proper operation of the development environment. If all three LED's do not light up, see the *HP 64120A Instrumentation Cardcage Service Manual* for information on correcting any problems.

Notes

Getting Started

Overview

This chapter tells you how to:

- Create a subdirectory in which you can store 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 pmon softkeys.
- Modify the default emulation configuration and map memory by loading a configuration file.
- Run an emulation session.

Introduction

This chapter gives an overview of the emulation process. Through example, it shows what you must do to prepare your system for emulation and how to make simple measurements. Work all exercises in the order presented. Then you will understand basic emulator operation.

Emulation System Used For Examples

The examples in this manual were developed with an emulation system that includes the components listed below.

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

If you do not have the Cross Assembler/Linker and C Cross Compiler specified above, you can still do the procedures in this chapter. Executable forms of the demonstration programs are supplied with your product software.

Make 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 these rules:

- The subdirectory name must have from one to fourteen characters. If it has more than fourteen characters, 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. For example, "FILENAME" is different from "filename."

Note

The path **/usr/hp64000/bin** must be added to the PATH parameter in your ".profile" file 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, for example, **/usr/hp64000/bin/pmon** instead of **pmon**.

Do the following 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 ".profile" file, add the line "cd $HOME/em68030" to your ".profile" file. You will then be in the new subdirectory when 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.

Initialize And Configure Your Measurement System

Note

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

See 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 an overview of the initialization and configuration process.

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

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 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 exits the measurement configuration mode and returns 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).

Prepare 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 by the program. 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. Chapter 4 describes memory mapping. This manual doesn't describe the assembly and compile procedures. See your assembler/linker/librarian and compiler manuals for detailed instructions.

Note

Executable files with names ending in ".X" (capital X) were created in HP-OMF format. If you installed your HP 64430 emulation software as an update, or if you intend to use executable files in IEEE format, remove the HP-OMF format absolute files from the demo directory with the command:

```
rm /usr/hp64000/demo/emul32/hp64430/*.X
```

Be sure to use capital X, not a small x, in the above command.

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, "Create the Absolute File In Your 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 "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 define a virtual system by mapping the 68030 MMU.

If you have a system printer, you can print the simint.c and towers.c demonstration programs with the command:

```
lp towers.c simint.c
```

Or, you can look at the files on-line with the commands:

```
more towers.c
more simint.c
```

Note

The README file in the demo directory tells more about the demonstration files and their use. To view the README file, enter the command:

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

Create the Absolute File In Your Subdirectory

The demonstration programs are installed in the directory **/usr/hp64000/demo/emul32/hp64430**. Copy these programs to your subdirectory by using the command:

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

Now enter your subdirectory by using the following command:

```
cd em68030
```

There is a makefile in the demonstration directory. Use the following commands to have the makefile create the absolute file for the getting started procedure:

```
make clean
```

Your display will show some 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 set of absolute files in your directory by entering the command:

```
make
```

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 Warning messages are normal. They involve a source file that won't be used in this manual.

When execution of the make file is complete, go to the paragraph titled "Preparing The Emulation System."

Use the Absolute File In The Demo Directory

If you do not have the HP compiler and linker specified earlier, you can use the absolute file in the directory **/usr/hp64000/demo/emul32/hp64430**. To use this absolute file, you must change to that directory. (If you copied the file to your directory, the pathnames in the SRU symbol database would be incorrect.) To change directories, press the **chng_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** (print working directory) command.

Prepare the Emulation System

To prepare the emulation system, you do the following:

1. Normally, you might connect the emulator probe into your target system (for in-circuit emulation). You will not do this for the getting started procedure in this chapter.

2. Access the emulator through the MEAS_SYS application.

3. Modify the default emulator configuration to match your system requirements. You can do this by using the **modify configuration** command or by loading a configuration file. You'll use the second method in this chapter.

4. Load 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). Chapter 6 discusses target system plug-in issues.

Access 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.

Modify the Default Emulation Configuration

When you start the emulator software, the default emulation configuration is loaded. You need to modify this configuration to create one that supports your demonstration programs. The demonstration directory contains a configuration file to make these modifications for you. Load the demonstration configuration file using the command:

load configuration config **Return**

Figure 3-1 lists the demonstration configuration file. The configuration file defines the emulation memory map, and answers the emulation configuration questions.

```
################################################################################
#
# This is the configuration file for the HP 64430 68030 Emulator/Analyzer
# demonstration software.
# If blocks of memory are mapped noncontiguously 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 07fffffffH emulation ram asynchronous width32
### Map 88k memory for all data sections.
map 0fffea000H thru 0ffffffffH 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
```

Figure 3-1. Demonstration Configuration File

```
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
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)

When the emulator finishes loading the memory mapper and background monitor, the STATUS line will show that the emulation processor is Reset. The emulator is ready for use.

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 that you can edit on your host system.

Load Emulation Memory

You are ready to begin an emulation session. Before you make any measurements, you must load memory with your program. To load emulation memory with the demo program, enter the command:

load memory `towers`

Use the Emulator

This section shows some simple emulator commands. Work through the examples in the sequence shown here. Otherwise, the displays you see may not be the same as those shown in the manual. After you have done the examples, you can execute other commands to understand 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 the emulator's analysis features.

Note

The displays you see on your system may be different from those shown in this manual. This depends on the type of terminal or workstation you are using.

Display The Source File

The **display source_file** command shows the source file on screen. Enter the following command to see the source file of the demonstration program.

display source_file towers.c Return

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

```
 Source File :
  file = towers.c
/*   LSD:@(#) 0.02 01Feb91
/*   @(mktid) (A.01.10 01Feb91)
/*                                                                  */
/*   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"         */

STATUS:   M68030--Reset_________________________________________________........
 display  source_file  towers.c


  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 source file. You can also move to a new area of the source file by using the **line_number** token for the **display source_file** command. Even though no line numbers appear in the source file listing, the source file display will reposition to place the line number you specify at the top of the screen.

Symbol Handling

When you load a program for the first time, the emulator uses the Symbolic Retrieval Utilities (SRU) to build a symbol database for each module. This database associates symbol names and symbol type information (not data types) with logical addresses. You will see a message on screen showing the module for which the database is being built.

Once a symbol database is created for a particular module, it does not need to be rebuilt unless the module is changed. You can rebuild modules using the **srubuild** utility (see the *HP 64000-UX System User's Guide*). If you reenter emulation without building symbols, the emulator software automatically rebuilds portions of the symbol database as you reference symbols in modified modules.

Note

You must use the -a and -p options when using srubuild to ensure proper handling of 68030 addresses. For example:

```
srubuild -a 68030 -p 64430 [options]
<filename> [buildfile]
```

Otherwise, srubuild will assume the object file is a 68000 object file, and will incorrectly truncate addresses.

Global symbol information is immediately available for the file that you loaded. To obtain local symbol information, you need to specify the module that contains the symbols.

You can use the symbol names instead of addresses when entering expressions as part of an emulation command. Therefore, you don't have to remember segment:offset information to make a measurement. Also, the emulator can display symbols as part of a measurement, using the **set symbols on** command. This helps you relate the measurement to your original program.

The 68030 emulator can read absolute files in HP-OMF or IEEE-695 format. For more information on SRU, refer to the *HP 64000-UX System User's Guide*. Additional information on symbol entry syntax is in the **--SYMB--** syntax pages of the *68030 Emulator Reference*.

When you load an absolute file into memory (unless you use the "nosymbols" syntax), symbol information is also loaded. You can display global symbols and symbols that are local to a source file.

You can set the current working symbol using the cws command.

```
cws <symbol>
```

To see the name of the current working symbol, type:

```
pws <symbol>
```

Displaying Global Symbols

The **display global_symbols** command displays global (externally defined) symbols in the program modules you have loaded in memory. To display global symbols, enter the command:

display global_symbols

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

```
Global symbols in towers
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
 __fflush                         00005B00 - 00005BB7 libc                  00F2
 _bufsync                         00005ECE - 00005F0B libc                  04C0
 _dbl_to_str                      00003724 - 00003C11 libc                  01E8
 _doprnt                          00003F38 - 00004F79 libc                  00AC
 _doscan                          00004FD2 - 0000528D libc                  0000
 _exec_funcs                      00001E82 - 00001EA1 libc                  0032
 _filbuf                          000058E4 - 00005A0D libc                  0000
 _findbuf                         00005D82 - 00005E23 libc                  0374
 _flsbuf                          00005BB8 - 00005CE5 libc                  01AA
 _memccpy                         000061DC - 00006207 libc                  0000
 _startup                         000009AE - 00000AE5 env                   0000
 _wrtchk                          00005E24 - 00005ECD libc                  0416
 _xflsbuf                         00005CE6 - 00005D81 libc                  02D8
 atexit                           00001E50 - 00001E81 libc                  0000
 atof                             00002B3E - 00002BC3 libc                  0C9A

STATUS:   M68030--Reset_________________________________________________........
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 of a file or module using the **display local_symbols_in** command. Enter the following command to view the demonstration program's local symbols:

display local_symbols_in towers

Note

If you were working with files compiled in HP-OMF format (the files with ".X" file name extensions in the demo directory), you would need to specify **towers.c**: in the above command. You use the "**.c**" file extension to specify C language files and the "**.s**" file extension to specify assembly language files.

An equivalent command is:

display local_symbols_in towers(module)

```
Symbols in towers(module)
Procedure symbols
Procedure name ________________   Address range __ Segment _____________   Offset
ask_for_number                    0000120C - 00001389 prog                    00B0
init_display                      00001610 - 000016C5 prog                    04B4
main                              00001162 - 00001205 prog                    0006
move_disc                         00001582 - 00001609 prog                    0426
pause                             00001390 - 000013C5 prog                    0234
place_disc                        00001524 - 0000157B prog                    03C8
remove_disc                       000014D0 - 0000151D prog                    0374
show_discs                        000013CC - 000014C9 prog                    0270
towers                            000016CC - 00001743 prog                    0570

Static symbols
Symbol name ____________________  Address range __ Segment _____________   Offset
blank_disc                        FFFEA334 - FFFEA343 data                    0184
disc_level                        FFFEA1C8 - FFFEA1E3 data                    0018
disc_word                         FFFEA344 - FFFEA3B3 data                    0194

STATUS:   M68030--Reset______________________________________________........
display local_symbols_in towers


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

To display the local symbols in the simint.c module, which was linked with towers.c to form the executable, type:

display local_symbols_in simint

```
Symbols in simint(module)
Procedure symbols
Procedure name _________________ Address range __ Segment _____________ Offset
disable_int                      00000050 - 00000061 simint                  0020
enable_int                       00000036 - 00000049 simint                  0006
sim_int_handler                  00000068 - 00000081 simint                  0038

Static symbols
Symbol name ____________________ Address range __ Segment _____________ Offset
sim_int_ca                       FFFEA1AC - FFFEA1AF data                    0004
sim_ints_serviced                FFFEA1A8 - FFFEA1AB data                    0000
trap14                           000000B8 - 000000BB                         0000

Filename symbols
Filename ______________________________________________________________________
simint.c


STATUS:    M68030--Reset_________________________________________________........
 display local_symbols_in simint


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

Line number symbols are available in the IEEE-695 file format. To display these, type:

display local_symbols_in towers."towers.c":

```
Symbols in towers(module)."towers.c":
Source reference symbols
Line range ____________________ Address range __ Segment _____________ Offset
#1-#128                        00001162 - 00001171 prog                 0006
#129-#130                      00001172 - 00001177 prog                 0016
#131-#133                      00001178 - 0000117B prog                 001C
#134-#135                      0000117C - 00001181 prog                 0020
#136-#137                      00001182 - 0000118B prog                 0026
#138-#139                      0000118C - 0000118F prog                 0030
#140-#140                      00001190 - 00001197 prog                 0034
#141-#141                      00001198 - 0000119B prog                 003C
#142-#142                      0000119C - 0000119D prog                 0040
#143-#144                      0000119E - 000011B1 prog                 0042
#145-#146                      000011B2 - 000011B5 prog                 0056
#147-#150                      000011B6 - 000011C9 prog                 005A
#151-#152                      000011CA - 000011DF prog                 006E
#153-#158                      000011E0 - 000011E1 prog                 0084
#159-#159                      000011E2 - 000011E3 prog                 0086

STATUS:   M68030--Reset_______________________________________________........
 display local_symbols_in towers."towers.c":


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

The SRU enforces symbol hierarchy. For example, there is a symbol named **towers** which is a module, and another symbol named **towers** which is a procedure in that module. To display the symbols for the **towers** procedure, type:

display local_symbols_in `towers.towers`

```
Symbols in towers(module).towers(procedure)
Procedure special symbols
Procedure special name __________ Address range __ Segment _____________ Offset
ENTRY                               000016CC          prog                     0570
EXIT                                00001742          prog                     05E6
TEXTRANGE                           000016CC - 00001743 prog                   0570









STATUS:   M68030--Reset__________________________________________________........
 display local_symbols_in towers.towers


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

Another way to specify this is by typing:

display local_symbols_in
`towers(module).towers(procedure)`

You can display local symbols for any symbol displayed in the global symbol listing. You cannot display symbol information for symbols that are created dynamically on the stack during runtime. For example, the procedure **towers** in **towers.c** has a variable called **ret_val**. The command

display local_symbols_in
`towers.towers.ret_val`

will fail, since **ret_val** is dynamically created. But, if you declare **ret_val** to be a static variable, the build process assigns a specific address location to this variable. SRU can then access this variable as a local symbol subordinate to the tower procedure in **towers.c**.

Display 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***

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

```
 Memory   :mnemonic :file = towers(module)."towers.c":
   address        data
  00001162       4E560000    LINK.W        A6,#$0000
  00001166       2F0B        MOVE.L        A3,-(A7)
  00001168       2F0A        MOVE.L        A2,-(A7)
  0000116A       45ED800C    LEA           ($800C,A5),A2
  0000116E       47FA0220    LEA           ($0220,PC),A3
  00001172       4EB90000+   JSR           $00000036
  00001178       42AD8008    CLR.L         ($8008,A5)
  0000117C       60000068    BRA.W         $000011E6
  00001180       4E71        NOP
  00001182       48780001    PEA           $00000001
  00001186       4EB80DEE    JSR           $00000DEE
  0000118A       588F        ADDQ.L        #4,A7
  0000118C       42AD8010    CLR.L         ($8010,A5)
  00001190       42A7        CLR.L         -(A7)
  00001192       4EBA047C    JSR           ($047C,PC)
  00001196       588F        ADDQ.L        #4,A7

STATUS:   M68030--Reset________________________________________________........
display memory main mnemonic


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

Adding Symbols To The Memory Display

You can use the **set symbols** command to obtain memory display that shows address values in terms of symbols defined in the towers.c source file. Enter the following command:

set symbols on Return

```
Memory  :mnemonic :file = towers(module)."towers.c":
  address       label          data
  00001162      |towers.main   4E560000    LINK.W        A6,#$0000
  00001166                     2F0B        MOVE.L        A3,-(A7)
  00001168                     2F0A        MOVE.L        A2,-(A7)
  0000116A                     45ED800C    LEA           ($800C,A5),A2
  0000116E                     47FA0220    LEA           (pro|towers.pause,PC),A3
  00001172                     4EB90000+   JSR           simin.enable_int
  00001178                     42AD8008    CLR.L         ($8008,A5)
  0000117C                     60000068    BRA.W         prog|main+$0084
  00001180                     4E71        NOP
  00001182                     48780001    PEA           :mon_stub+$0001
  00001186                     4EB80DEE    JSR           sys.clear_screen
  0000118A                     588F        ADDQ.L        #4,A7
  0000118C                     42AD8010    CLR.L         ($8010,A5)
  00001190                     42A7        CLR.L         -(A7)
  00001192                     4EBA047C    JSR           (tow.init_display,PC)
  00001196                     588F        ADDQ.L        #4,A7

STATUS:   M68030--Reset__________________________________________________........
 set  symbols  on


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

Note

Symbols can also be displayed when you are viewing memory formats of absolute, binary, and real, as well as mnemonic.

Adding Source-File Lines To The Memory Display

You can use the **set source** command to obtain memory displays that show lines of source-file content preceding the assembly language code it emitted. Enter the following command:

set source on Return

```
 Memory  :mnemonic :file = towers(module)."towers.c":
   address      label         data
     124        static void towers();
     125        static int  ask_for_number();
     126
     127        main()
     128        {
  00001162      |towers.main  4E560000    LINK.W        A6,#$0000
  00001166                    2F0B        MOVE.L        A3,-(A7)
  00001168                    2F0A        MOVE.L        A2,-(A7)
  0000116A                    45ED800C    LEA           ($800C,A5),A2
  0000116E                    47FA0220    LEA           (pro|towers.pause,PC),A3
     129        #ifdef INTERRUPTS
     130             enable_int();
  00001172                    4EB90000+   JSR           simin.enable_int
     131        #endif
     132
     133             run_continuous = FALSE;

STATUS:    M68030--Reset__________________________________________________........
 set  source  on


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

Modify 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**. You will set the value of **loc_delay** to 0 so that the program runs at maximum speed. To watch the memory display change as the variable is modified, you will display an area in memory repetitively, then modify the memory. Enter the following command:

display memory `loc_delay` ***long repetitively***

You should see a display similar to the following on your workstation screen. The loc_delay variable is the first long word on the screen. Its value is 000001F4 in the illustration.

```
 Memory  :long words :blocked :repetitively
   address           data       :hex                                    :ascii
  00001744-50        000001F4 09095075 7A7A6C65 20776974      ......Pu  zzle wit
  00001754-60        68202564 20646973 63732063 616E2062      h %d dis  cs can b
  00001764-70        6520736F 6C766564 20696E20 2564206D      e solved   in %d m
  00001774-80        6F766573 2E202020 20200A00 0A0A4578      oves.         ....Ex
  00001784-90        65637574 6520276D 6F646966 79206B65      ecute 'm  odify ke
  00001794-A0        79626F61 72645F74 6F5F7369 6D696F27      yboard_t  o_simio'
  000017A4-B0        20746865 6E20656E 74657220 6F6E6520       then en  ter one
  000017B4-C0        6F662074 68652066 6F6C6C6F 77696E67      of the f  ollowing
  000017C4-D0        3A0A094E 756D6265 72206F66 20646973      :..Numbe  r of dis
  000017D4-E0        63732074 6F207573 65205B31 2D25645D      cs to us  e [1-%d]
  000017E4-F0        0A092730 2720746F 20657869 74207072      ..'0' to   exit pr
  000017F4-00        6F677261 6D0A0927 43272074 6F207275      ogram..'  C' to ru
  00001804-10        6E20636F 6E74696E 756F7573 6C792075      n contin  uously u
  00001814-20        73696E67 206C6173 74206E75 6D626572      sing las  t number
  00001824-30        20656E74 65726564 0A0A003F 00256400       entered  ...?.%d.
  00001834-40        20696E76 616C6964 20726570 65617420       invalid   repeat

STATUS:    M68030--Reset____________________________________________........
 display   memory loc_delay long repetitively


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

Enter the command:

modify memory long loc_delay ***to*** 0

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
  00001744-50        00000000 09095075 7A7A6C65 20776974     ......Pu  zzle wit
  00001754-60        68202564 20646973 63732063 616E2062     h %d dis  cs can b
  00001764-70        6520736F 6C766564 20696E20 2564206D     e solved   in %d m
  00001774-80        6F766573 2E202020 20200A00 0A0A4578     oves.        ....Ex
  00001784-90        65637574 6520276D 6F646966 79206B65     ecute 'm  odify ke
  00001794-A0        79626F61 72645F74 6F5F7369 6D696F27     yboard_t  o_simio'
  000017A4-B0        20746865 6E20656E 74657220 6F6E6520      then en  ter one
  000017B4-C0        6F662074 68652066 6F6C6C6F 77696E67     of the f  ollowing
  000017C4-D0        3A0A094E 756D6265 72206F66 20646973     :..Numbe  r of dis
  000017D4-E0        63732074 6F207573 65205B31 2D25645D     cs to us  e [1-%d]
  000017E4-F0        0A092730 2720746F 20657869 74207072     ..'0' to   exit pr
  000017F4-00        6F677261 6D0A0927 43272074 6F207275     ogram..'  C' to ru
  00001804-10        6E20636F 6E74696E 756F7573 6C792075     n contin  uously u
  00001814-20        73696E67 206C6173 74206E75 6D626572     sing las  t number
  00001824-30        20656E74 65726564 0A0A003F 00256400      entered  ...?.%d.
  00001834-40        20696E76 616C6964 20726570 65617420      invalid   repeat

STATUS:    M68030--Reset_____________________________________________........
 modify  memory  long loc_delay to 0


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

Run From The Transfer Address

You have used some of the display and modify features of the emulator. Now you are ready to run the demonstration program and try some run-time features. Enter the command:

run from transfer_address

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

Display Registers

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

display registers cpu

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

program counter (NextPC)
source function code register (SFC)
destination function code register (DFC)
data registers (D0—D7)
address registers (A0—A6)
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)

```
M68030 Registers

 NextPC 00000C18      SFC 0 MOT RSV 0            DFC 0 MOT RSV 0
  D0-D7 00000092 00000001 00000092 000058E4 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA057 FFFEA484 FFFEA574 FFFEA054 FFFF21A8 7FFFFDF0
    USP 7FFFFFF8               1 t0 s  m  i  x  n  z  v .c     CAAR 00000000
    MSP 7FFFFFF8   STATUS 2708  0  0  1  0  7  0  1  0  0  0       VBR 00000000
   *ISP 7FFFFDEC                 wa dbe  fd  ed ibe  fi  ei
                     CACR 0000    0   0   0   0   0   0   0









STATUS:   M68030--Running_______________________________________________........
 display  registers  cpu


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

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 occurs.

```
M68030 Registers

 NextPC 00000C18      SFC 0 MOT RSV 0           DFC 0 MOT RSV 0
  D0-D7 00000092 00000001 00000092 000058E4 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA057 FFFEA484 FFFEA574 FFFEA054 FFFF21A8 7FFFFDF0
    USP 7FFFFFF8               1 t0 s  m  i  x  n  z  v  c     CAAR 00000000
    MSP 7FFFFFF8   STATUS 2708  0  0  1  0  7  0  1  0  0  0       VBR 00000000
   *ISP 7FFFFDEC                 wa dbe  fd  ed ibe  fi  ei
                     CACR 0000    0   0   0   0   0   0   0

 NextPC 00000C18      SFC 0 MOT RSV 0           DFC 0 MOT RSV 0
  D0-D7 00000092 00000001 00000092 000058E4 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA057 FFFEA484 FFFEA574 FFFEA054 FFFF21A8 7FFFFDF0
    USP 7FFFFFF8               1 t0 s  m  i  x  n  z  v  c     CAAR 00000000
    MSP 7FFFFFF8   STATUS 2708  0  0  1  0  7  0  1  0  0  0       VBR 00000000
   *ISP 7FFFFDEC                 wa dbe  fd  ed ibe  fi  ei
                     CACR 0000    0   0   0   0   0   0   0


STATUS:    M68030--Running in monitor____________________________________........
 break


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

Use 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

The register display is updated after each step. The second entry on the display shows an additional line. 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 shows the register activity for each executed instruction.

```
M68030 Registers

 NextPC 00000C18      SFC 0 MOT RSV 0           DFC 0 MOT RSV 0
  D0-D7 00000092 00000001 00000092 000058E4 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA057 FFFEA484 FFFEA574 FFFEA054 FFFF21A8 7FFFFDF0
    USP 7FFFFFF8               1 t0 s  m  i  x  n  z  v  c     CAAR 00000000
    MSP 7FFFFFF8   STATUS 2708  0  0  1  0  7  0  1  0  0  0       VBR 00000000
   *ISP 7FFFFDEC                 wa dbe  fd  ed ibe  fi  ei
                     CACR 0000   0   0   0   0   0   0   0

     PC 00000400   Opcode LEA           :TopOfStack,A7                 4FF98000
 NextPC 00000406      SFC 0 MOT RSV 0           DFC 0 MOT RSV 0
  D0-D7 00000092 00000001 00000092 000058E4 000000FF 00000000 00000064 00000000
  A0-A6 000000FF FFFEA057 FFFEA484 FFFEA574 FFFEA054 FFFF21A8 7FFFFDF0
    USP 7FFFFFF8               1 t0 s  m  i  x  n  z  v  c     CAAR 00000000
    MSP 7FFFFFF8   STATUS 2708  0  0  1  0  7  0  1  0  0  0       VBR 00000000
   *ISP 80000000                 wa dbe  fd  ed ibe  fi  ei
                     CACR 0000   0   0   0   0   0   0   0

STATUS:   M68030--Running in monitor____________________________________........
 step  from  transfer_address


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

Enter the command:

step

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

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

step 25

Notice that the screen is updated with register information each time a program step is executed. While the **step** command is executing, 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 the screen.

Stepping Through The Program In Memory

You can use the **step** function to watch each of your program instructions as they execute. Enter the commands:

display memory main **mnemonic Return**

set source off Return

step from main **Return**

Your display should appear as in the following illustration.

```
Memory  :mnemonic :file = towers(module)."towers.c":
  address      label          data
 00001162      |towers.main  4E560000    LINK.W        A6,#$0000
 00001166                    2F0B        MOVE.L        A3,-(A7)
 00001168                    2F0A        MOVE.L        A2,-(A7)
 0000116A                    45ED800C    LEA           ($800C,A5),A2
 0000116E                    47FA0220    LEA           (pro|towers.pause,PC),A3
 00001172                    4EB90000+   JSR           simin.enable_int
 00001178                    42AD8008    CLR.L         ($8008,A5)
 0000117C                    60000068    BRA.W         prog|main+$0084
 00001180                    4E71        NOP
 00001182                    48780001    PEA           :mon_stub+$0001
 00001186                    4EB80DEE    JSR           sys.clear_screen
 0000118A                    588F        ADDQ.L        #4,A7
 0000118C                    42AD8010    CLR.L         ($8010,A5)
 00001190                    42A7        CLR.L         -(A7)
 00001192                    4EBA047C    JSR           (tow.init_display,PC)
 00001196                    588F        ADDQ.L        #4,A7

STATUS:   M68030--Running in monitor____________________________________........
 step  from  main


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

The highlighted line on your screen (not highlighted on the illustration above) is the "next PC" line indicated by the program counter of the emulation processor. Enter the command:

step Return

The former "next PC" instruction was executed and the new next PC is now highlighted on screen.

When the "next PC" is in a portion of memory that is presently off screen, the display window shifts to show it. Enter the command:

step 4 **Return**

```
 Memory  :mnemonic :file = simint(module)."simint.c":
   address      label          data
  00000036     s.enable_int   4E560000    LINK.W        A6,#$0000
  0000003A                    70FF        MOVEQ         #$FFFFFFFF,D0
  0000003C                    2B408004    MOVE.L        D0,($8004,A5)
  00000040                    4E71        NOP
  00000042                    4E5E        UNLK          A6
  00000044                    4E71        NOP
  00000046                    4E71        NOP
  00000048                    4E75        RTS
  0000004A                    4E71        NOP
  0000004C                    4E71        NOP
  0000004E                    4E71        NOP
  00000050     .disable_int   4E560000    LINK.W        A6,#$0000
  00000054                    42AD8004    CLR.L         ($8004,A5)
  00000058                    4E71        NOP
  0000005A                    4E5E        UNLK          A6
  0000005C                    4E71        NOP

STATUS:   M68030--Running in monitor____________________________________........
 step 4

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

The last step shifted the display window beyond the instructions that had been on screen. The new next PC is shown at the top of the display (highlighted on your screen, but not in the above illustration).

Trace Processor Activity

The trace function 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, see 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
```

This traces all activity on the bus for 1023 bus cycles before and 1023 bus cycles after the address labeled "main" occurs. The STATUS line will display "Trace in process." Enter the command:

```
run from main
```

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

```
display trace
```

The trace list is displayed with the trigger state shown 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 (towers+00000004h).

```
Trace List                                              Mode:logical address
Label:       Address                          Opcode or Status                  time count
Base:        symbols                        mnemonic w/symbols                   relative
-0007    sysstac+00007FEA   $----0000       supr data long wr log addr    10.8us
-0006    sysstac+00007FEC   $1162----       supr data word wr log addr    0.80us
-0005    sysstac+00007FEE   $----007C       supr data word wr log addr    10.8us
-0004    sysstac+00007FE8   $2708----       supr data word rd log addr    19.8us
-0003    sysstac+00007FEE   $----007C       supr data word rd log addr    0.44us
-0002    sysstac+00007FEA   $----0000       supr data long rd log addr    0.44us
-0001    sysstac+00007FEC   $1162----       supr data word rd log addr    0.44us
trigger  |towers+00000004   $4E714E56       supr prgm long rd log addr    0.68us
+0001    pr|main+00000002   $00002F0B       supr prgm long rd log addr    0.48us
+0002    pr|main+00000006   $2F0A45ED       supr prgm long rd log addr    0.52us
+0003    sysstac+00007FEC   $7FFFFFFC       supr data long wr log addr    0.44us
+0004    pr|main+0000000A   $800C47FA       supr prgm long rd log addr    0.44us
+0005    sysstac+00007FE8   $00001390       supr data long wr log addr    0.44us
+0006    sysstac+00007FE4   $FFFEA1B4       supr data long wr log addr    0.56us
+0007    pr|main+0000000E   $02204EB9       supr prgm long rd log addr    0.44us

STATUS:   M68030--Running                     Trace complete________________........
 display  trace


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

The address is displayed in terms of source-file symbols plus hexadecimal offsets. The data values in the default trace list are displayed as hexadecimal numbers. The emulator also can display values in assembly language mnemonics with symbols. Enter the commands:

display trace disassemble_from_line_number 0

Note that in the updated trace display, the trigger line (line 0) is the first line in the trace display. Address towers+00000004h corresponds to the **main** label in the demonstration program. The instruction **LINK.W** is the second instruction in the long word at the trigger address in the example program.

```
Trace List                                        Mode:logical address
Label:       Address                   Opcode or Status                  time count
Base:        symbols                 mnemonic w/symbols                   relative
trigger |towers+00000004 NOP                                               0.68us
       =prog|towers.main LINK.W        A6,#$0000
+0001  =pr|main+00000004 MOVE.L        A3,-(A7)                            0.48us
+0002   pr|main+00000006 MOVE.L        A2,-(A7)                            0.52us
       =pr|main+00000008 LEA           ($800C,A5),A2
+0003   sysstac+00007FEC   $7FFFFFFC     supr data long wr log addr        0.44us
+0004  =pr|main+0000000C LEA           (pro|towers.pause,PC),A3            0.44us
+0005   sysstac+00007FE8   $00001390     supr data long wr log addr        0.44us
+0006   sysstac+00007FE4   $FFFEA1B4     supr data long wr log addr        0.56us
+0007  =pr|main+00000010 JSR           simin.enable_int                    0.44us
+0008   pr|main+00000012   $00000036     supr prgm long rd log addr        0.56us
+0009    |simint+00000004 NOP                                              0.64us
       =simin.enable_int LINK.W        A6,#$0000
+0010   sysstac+00007FE0   $00001178     supr data long wr log addr        0.48us
+0011  =enable_+00000004 MOVEQ         #$FFFFFFFF,D0                       0.44us

STATUS:   M68030--Running                  Trace complete________________........
 display  trace  disassemble_from_line_number 0


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

The preceding trace list shows the order of activity that occurred on the emulation bus. When the processor fetches an instruction and places it in its queue, that instruction is captured and placed in trace memory. Before the execution of that instruction, there will be bus cycles from the execution of instructions that were fetched earlier and placed in the queue. There may be additional fetches of instructions to be executed later. Finally, you will see the bus cycles from the execution of the instruction you were watching. The effects of the processor queue makes trace lists hard to read.

The analysis software has an option that replaces bus cycle execution with a logical view of program execution. Enter the command:

display trace disassemble_from_line_number 0 **dequeued Return**

```
Trace List                                              Mode:logical address
Label:      Address                      Opcode or Status                    time count
Base:       symbols                     mnemonic w/symbols                    relative
trigger |towers+00000004 NOP                                                   0.68us
       =prog|towers.main LINK.W        A6,#$0000
       =sysstac+00007FEC  stck sdata wr:$7FFFFFFC
+0001  =pr|main+00000004 MOVE.L        A3,-(A7)                                0.48us
       =sysstac+00007FE8  dest sdata wr:$00001390
+0002   pr|main+00000006 MOVE.L        A2,-(A7)                                0.52us
       =sysstac+00007FE4  dest sdata wr:$FFFEA1B4
       =pr|main+00000008 LEA           ($800C,A5),A2
+0004  =pr|main+0000000C LEA           (pro|towers.pause,PC),A3                0.88us
+0007  =pr|main+00000010 JSR           simin.enable_int                        1.44us
       =sysstac+00007FE0  stck sdata wr:$00001178
+0009  =simin.enable_int LINK.W        A6,#$0000                               1.20us
       =sysstac+00007FDC  stck sdata wr:$7FFFFFEC
+0011  =enable_+00000004 MOVEQ         #$FFFFFFFF,D0                           0.92us
+0012   enable_+00000006 MOVE.L        D0,($8004,A5)                           0.48us

STATUS:   M68030--Running                Trace complete________________........
 display  trace  disassemble_from_line_number 0 dequeued


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

The above trace list aligns instructions with their operands, suppresses the display of unused prefetches, and adds information to clarify the execution of traced code.

You also can display the source program lines that correspond to the assembly code in the trace list. Enter the command:

display trace source on inverse_video on

The display is updated with the source code line displayed in inverse video immediately before the related assembly code. (Inverse video is not shown in this manual illustration.)

```
Trace List                                        Mode:logical address
Label:       Address          Opcode or Status w/ Source Lines       time count
Base:        symbols                 mnemonic w/symbols               relative
trigger |towers+00000004 NOP                                            0.68us
       #########towers.c - line     1 thru    128 ##############################
       static void towers();
       static int  ask_for_number();

       main()
       {
       =prog|towers.main LINK.W       A6,#$0000
       =sysstac+00007FEC  stck sdata wr:$7FFFFFFC
+0001  =pr|main+00000004 MOVE.L       A3,-(A7)                          0.48us
       =sysstac+00007FE8  dest sdata wr:$00001390
+0002   pr|main+00000006 MOVE.L       A2,-(A7)                          0.52us
       =sysstac+00007FE4  dest sdata wr:$FFFEA1B4
       =pr|main+00000008 LEA          ($800C,A5),A2
+0004  =pr|main+0000000C LEA          (pro|towers.pause,PC),A3          0.88us

STATUS:   M68030--Running                 Trace complete________________........
 display  trace   source  on  inverse_video  on


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

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

Use Software Breakpoints

The **modify software_breakpoints** command lets you set software breakpoints in your program code. This useful feature lets you break program execution at the point you select. You can then examine register values, display or modify memory locations, and perform other operations before continuing execution of your program.

When you set a breakpoint, the emulator replaces the code at the memory location you specify with a 68030 BKPT instruction. Enter the command:

break

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

modify software_breakpoints set one_shot `towers.ask_for_number`

The emulator replaces the instruction at the address referenced 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 command:

display memory `towers.ask_for_number` ***mnemonic***

Note

Remember to use the command form "towers.c:_ask_for_number" above if you used the HP-OMF format demonstration program instead of the IEEE format file.

The display shows a **BKPT 7** instruction at address 0000120CH. The asterisk "*" shown to the left-hand side of this line indicates an active software breakpoint has been set for it.

```
 Memory  :mnemonic :file = towers(module)."towers.c":
   address      label         data
      165       /***** Towers routines *****/
      166
      167       static int ask_for_number(num)
      168       int *num;
      169       {
* 0000120C      ask_for_numb  484F        BKPT          #7
  0000120E                    000048E7    ORI.B         #$48E7,D0
  00001212                    3E38242E    MOVE.W        libc|atof+$058A,D7
  00001216                    0008200D    ORI.B         #$200D,A0
  0000121A                    0680FFFF+   ADDI.L        #$FFFF82DC,D0
  00001220                    2440        MOVEA.L       D0,A2
  00001222                    47FB8170+   LEA           (li|printf.printf,PC),A3
  0000122A                    49FB8170+   LEA           (libc|puts.puts,PC),A4
      170            char err_char1,err_char2;
      171            int last_num,ret_val;
      172

STATUS:   M68030--Running in monitor      Trace complete________________........
 display memory towers.ask_for_number mnemonic


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

Enter the command:

run from transfer_address

The demonstration program runs from the transfer_address (_main) until the BKPT instruction is executed. The BKPT instruction forces a break into the emulation monitor. The status line displays the message, "**STATUS: Software breakpoint hit at address = 120CH**".

The inverse video of the source lines runs together with the highlighting of the "next PC" line. To avoid confusion, enter the command:

set source on inverse_video off

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

```
 Memory  :mnemonic :file = towers(module)."towers.c":
   address      label          data
     165        /***** Towers routines *****/
     166
     167        static int ask_for_number(num)
     168        int *num;
     169        {
  0000120C      ask_for_numb  4E560000    LINK.W        A6,#$0000
  00001210                    48E73E38    MOVEM.L       D2-D6/A2-A4,-(A7)
  00001214                    242E0008    MOVE.L        ($0008,A6),D2
  00001218                    200D        MOVE.L        A5,D0
  0000121A                    0680FFFF+   ADDI.L        #$FFFF82DC,D0
  00001220                    2440        MOVEA.L       D0,A2
  00001222                    47FB8170+   LEA           (li|printf.printf,PC),A3
  0000122A                    49FB8170+   LEA           (libc|puts.puts,PC),A4
     170            char err_char1,err_char2;
     171            int last_num,ret_val;
     172

STATUS:   M68030--Running in monitor      Trace complete________________........
 set  source  on  inverse_video  off


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

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

run

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. Refer to 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

Your screen should look like 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________________........
display  simulated_io


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

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

modify keyboard_to_simio

The keyboard is now assigned to simulated I/O and is available 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:

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

               ||                       ||                       ||
               ||                       ||                       ||
              1||1                      ||                       ||
           4444||4444                   ||                       ||
          55555||55555                  ||                       ||
         666666||666666                 ||                     22||22
        7777777||7777777                ||                    333||333
        ----------------        ----------------        ----------------
             Peg  0                  Peg  1                  Peg  2

                 Solution for Towers with 7 discs.
                 Move #6:  Move disk 2 from peg 1 to peg 2
                 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 simulated I/O, see chapter 4 and the *Simulated I/O Operating Manual* supplied with your HP 64000-UX system.

Ending The Emulation Session

To end the emulation session, enter the command:

```
end release_system
```

The system will return to the **MEAS_SYS** application level.

This completes your introduction to the 68030 emulation system. You have used a make file to assemble and compile program modules, and link your program modules. Then you have used some some basic emulator features. 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 contains a series of commands that accomplish a particular function. Command files are ideal for initializing 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 enter your emulation session easily. Refer to the chapter "Creating and Using Command Files" in the *HP 64000-UX User's Guide* for detailed information on command files.

Use The DeMMUer

Use this procedure only if:

- You have a deMMUer in your emulation/analysis system hardware.
- Your system design uses the 68030 MMU.

This section shows how the deMMUer translates physical addresses that 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 *68030 Internal Analyzer User's Guide.*

The internal analyzer needs logical addresses to interpret 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 commands to reactivate the system and prepare it for this demonstration:

```
emul683k em68030
load configuration virtual
load memory physical os
```

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 defined in the file "os.s."

In this operating system script, physical memory maps 1:1 to logical memory. Thus, addresses do not change when the MMU is enabled. Mapping the memory 1:1 is not required for emulator operation, but simplifies this example. Enter the command:

```
set analysis mode logical
```

This command turns on the deMMUer so that it will supply logical addresses to the analyzer. 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 commands:

trace
display trace symbols on
run from entry(static)

The "entry" symbol used in the last command is a symbol in the data base of the "os" program. This starts the "os" program at the appropriate point. Your screen should look like 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   entry.reset       $00000EB0     supr prgm long rd log addr ------------
+0001    :entry+00000004    $00000A74     supr prgm long rd log addr    0.36us
+0002       000F0EF8        $2700----     supr data word wr log addr    406.ms
+0003       000F0EFA        $----000F     supr data long wr log addr    11.6us
+0004       000F0EFC        $0000----     supr data word wr log addr    0.72us
+0005       000F0EFE        $----0000     supr data word wr log addr    11.5us
+0006       000F0EF8        $2700----     supr data word rd log addr    21.1us
+0007       000F0EFE        $----0000     supr data word rd log addr    0.40us

STATUS:   M68030--Running                     Trace complete________________........
 run from entry(static)


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

Enter the commands:

break
load memory towers

The above command loads the towers program logically through the monitor. (This requires a few minutes.)

When the program has loaded, enter the following commands:

trace TRIGGER_ON a= long_aligned main
run from transfer_address

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
display trace source on inverse_video on

In the resulting display, 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 as it did before you used memory management. The deMMUer will translate addresses as needed 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 w/ Source Lines      time count
Base:        symbols                mnemonic w/symbols              relative
trigger |towers+00000004 NOP                                        0.40us
       #########towers.c - line     1 thru   128 ##############################
       static void towers();
       static int  ask_for_number();

       main()
       {
       =prog|towers.main LINK.W        A6,#$0000
+0001   sysstac+00007FC8   $00000ADA     supr data long wr log addr    0.40us
+0002  =pr|main+00000004 MOVE.L        A3,-(A7)                        0.40us
+0003   pr|main+00000006 MOVE.L        A2,-(A7)                        0.44us
       =pr|main+00000008 LEA           ($800C,A5),A2
+0004   sysstac+00007FC4   $7FFFFFF0     supr data long wr log addr    0.40us
+0005  =pr|main+0000000C LEA           (pro|towers.pause,PC),A3        0.40us
+0006   sysstac+00007FC0   $FFFEAD7C     supr data long wr log addr    0.40us

STATUS:   M68030--Running                Trace complete________________........
 display trace source on inverse_video on


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

End Of DeMMUer Demonstration

To end this demonstration (and the emulation session), enter the command:

end release_system

The system will return to the **MEAS_SYS** application level.

4

Answering Emulation Configuration Questions

Overview

This chapter:

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

Introduction

You configure the 68030 emulator 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 needs by answering a series of emulation configuration questions. After modifying the emulation configuration, you can save it to a file. You can then load this file each time you enter emulation.

Your answers to the configuration questions define:

- how the 68030 emulator is configured
- how resources are shared between the emulator and your target system
- how the emulator and target system interact
- what operations are enabled in the emulation environment.

The configuration questions cover the following emulation configuration items:

- 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. This chapter, uses the example names from chapter 3, "Getting Started." To run emulation, do the following:

1. Press **MEAS_SYS**.
2. Press **emul683k em68030 Return**.

Modify the Configuration File

To modify the configuration, enter the command:

```
modify configuration
```

A series of questions are displayed on your workstation screen. Your answers to these questions define the configuration of the emulation hardware and software for a specific application. Each question has a default response. This chapter shows additional options in parentheses. You select the default response 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 existing emulation configuration file, the default responses are the ones stored in that configuration file.

Note

If you need to return to a previous question, press the **RECALL** softkey. Each time you press **RECALL**, the emulator backs up by one configuration question. 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 a break into the 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 that 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, then resume execution of your program. In real-time run mode, you can execute any command that does not cause a break to the emulation monitor. Commands requiring target memory or register accesses are disabled when a user program is running. You can only execute these commands 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.

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

no — All selected emulator functions are enabled. The emulation system can break to the monitor whenever a command requiring breaks is executed.

yes — Target memory and register accesses are disabled when a user program is running.

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?	no
Enable emulator use of INT7?	no
Software BKPT instruction number (0..7)?	7
Default stack pointer for background?	0ffffffffh
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 are disabled, and their associated softkeys are 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)

Enable emulator use of the monitor? yes (no)

yes — All emulation commands and features implemented with the emulation monitor are enabled.

no — Configuration questions that refer to functions requiring the emulation monitor are not asked. If no emulation monitor is loaded, all commands and features requiring the emulation monitor are disabled and their associated softkeys are turned off. If you enable the MMU (in a later question), only direct physical memory accesses will succeed because logical address accesses are made through the monitor. The next question asked is "Modify memory configuration?"

Reset Into the Monitor

Note

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

The next question lets you select whether the emulation **reset** command resets the processor 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 doesn't affect reset signals generated by your target system.

Reset into the monitor? yes (no)

yes — The emulation reset command resets the processor 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.

Enable emulator use of INT7? yes (no)

The emulation break function uses the level 7 interrupt autovector (INT7) processor resource to interrupt the user program and enter the emulation monitor program. 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.

yes — All selected emulation functions are available.

no — All emulation break signals to the processor are disabled. The only ways to enter the monitor program are:

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

Note

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

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 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 the emulator can use the 68030 BKPT instructions for software breaks. The **modify software_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.

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

Defaulting the Stack Pointer For the Background Monitor

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

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

Select To Block ECS, OCS Signals During Background Monitor Cycles

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

yes — Blocks the ECS and OCS signals to the target system during background monitor execution.

no — Passes the ECS and OCS signals to the target system during background monitor execution. This makes the processor look as though it is executing out of cache.

Choose To Perform Periodic Foreground Accesses

This question determines whether the background monitor generates keepalive cycles. The keepalive function causes a periodic read cycle at a specified address. Some target systems need to see continuous cycles. When the background monitor is executing, AS, DS, and DBEN signals are blocked.

Note

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

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

Selecting Address for Periodic Foreground Access

Note

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

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

Enabling the Foreground Monitor

This question allows you to select the foreground monitor. The foreground monitor may be loaded by the background monitor. If you choose 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 user-defined coprocessor support. The foreground monitor does use target resources, and does not allow physical memory access when you enable the MMU in the configuration. If you answer **no** to this question, the sequence will skip to the "Modify memory configuration?" question.

```
Enable foreground monitor? no (yes)
```

Interlock or Provide Termination for the Foreground Monitor

This question allows you to choose 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.

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

Using Custom Coprocessors

Note

Custom register access is supported only with the foreground monitor enabled, except for MMU registers. Both foreground and background monitors support MMU registers display and modification.

The 68030 emulator can access floating point processors, memory management units, and other coprocessors in your target system. You can both display and modify coprocessor register sets. To use custom coprocessors with the emulator, you must provide a custom register format file defining the coprocessor register set. You also must modify the emulation monitor program as described in chapter 8. This must be done before you modify the emulation configuration.

Any custom registers? no (yes)

yes	The emulator can access the custom coprocessors that you have defined in a custom register format file.
no	Use of custom coprocessors is disabled.

Specifying The Custom Coprocessor File

Note

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

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

The default answer is NULL. The MMU is supported internally. There is an example custom register format file provided with your emulation software. The example is 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.

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 depends on your user program and on the configuration of your target system, if one is available. As your system design matures, your memory map requirements probably will change. As your requirements change, you will need to modify your configuration file.

The following questions let you review and modify the memory configuration.

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 — You can modify the memory map and deMMUer configuration. The current memory map is displayed. Memory configuration is explained in the following sections.

no — Skips the memory configuration. A **no** response configures memory as specified by the current emulation configuration file. If you select **no**, the next question is "Modify emulator pod configuration?"

Break on Write to ROM

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

yes — A break to the 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 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 (for example: **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

```
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.

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 processor memory space required for your applications must be mapped to emulation memory, target memory, or guarded memory. Emulation memory is memory that is in the emulator pod. Target memory is memory that is 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 cause an emulation memory break. The following error message is displayed:

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

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

For emulation to work correctly, the memory mapper must be programmed to correspond to emulation memory and target system memory resources. 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.

Note

The memory map specifies memory regions in physical address space, not logical address space. If your system uses the MMU, you must know which physical addresses are used.

During emulation, the memory mapper monitors the address bus and provides the attributes for the address present at any given time. The emulator hardware uses this information to control the data flow between the emulation processor and memory resources.

Memory Map Display Organization. Figure 4-1 shows the default memory map display. Each entry line shows the entry number, address range starting and ending values, 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 4-1. Default Memory Map Display

Softkey labels are displayed for the memory mapper commands. You can:

- add new map entries
- overlay existing map entries
- modify existing entries (including the default mapping attributes)
- modify the deMMUer configuration
- delete currently defined entries

- end the map definition session.

The following sections describe these commands.

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 five memory types.

Memory entries can be further defined by function code. Emulation memory also can 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 you enter are physical addresses.

The memory mapper has a resolution of 256 (0..ffH) bytes. When the mapper software processes the inputs, it rounds the entry range to integral multiples of 256 bytes. The final range includes all specified memory space, plus the remainder of any 256-byte blocks that were partially specified.

Note

If the end address of a specified address range is the same as the first address of a 256-byte memory block (for example: 100h, xxxxxx00h, and so on), the end address value is rounded down one byte (for example: to 0ffh, xxxxxxffh, and so on).

This can cause a problem if you 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
```

the error message "**ERROR: Lower address in range greater than upper address**" is displayed. The emulator rejects this command because the ending address (when rounded down to 0ffh) is less than the starting address (100h).

Emulation memory is displayed and loaded directly by the emulation software using the memory port assigned to the host processor. If you enable the MMU, logical addresses are loaded 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 by using the emulation monitor.

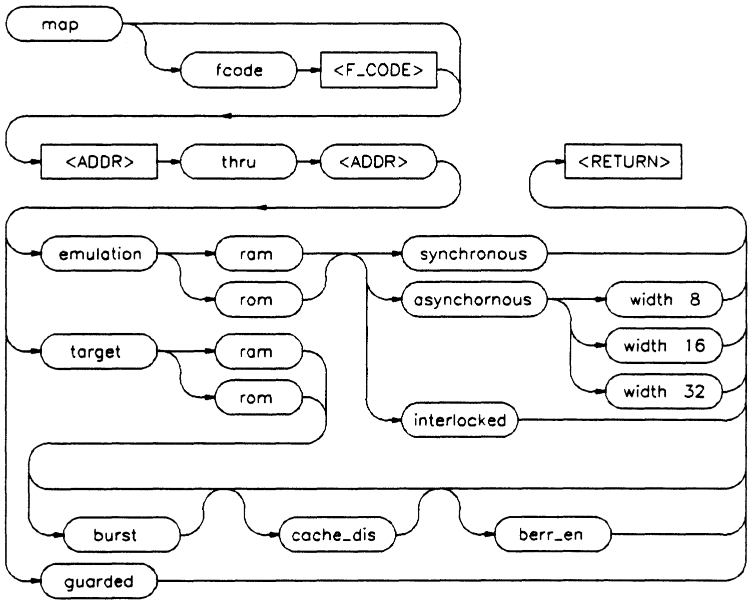
Multiple processor address ranges can be overlaid onto the same physical emulation memory by using the **map_overlay** command. Overlaying applies only to emulation memory. The emulator cannot overlay target system memory resources.

Emulation Monitor Program Memory Requirements. You must know some information about the emulation monitor (delivered as part of your emulator software package) before you link the monitor program and map memory space. Chapter 7 describes the emulation monitor, including memory requirements for the program. See the paragraphs titled "Emulation Monitor Memory Requirements" in chapter 7.

Using The Map Command

All memory map entries have an address range and attributes that specify the type of memory in the selected address range. A specific function code and address width (port size) can be assigned to a memory map entry. You map memory by 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 code. See the section in this chapter on the **modify defined_codes** command for more information.

<ADDR> — defines a pattern of up to 32 bits that specifies a particular memory location. That bit pattern can be entered as a binary, octal, hexadecimal, or decimal number.

guarded — designates an address range that is not expected to be accessed. Any processor access to a location within such a range will cause a break in program execution. No emulation memory is consumed by an address range specified as **guarded**.

emulation	designates memory supplied by the emulation system.
rom	designates memory that 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 that can be read from or written to without restriction.
interlocked	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 doesn't consume any emulation memory.
burst	enables the burst mode access.

cache_dis   disables caching for an address range.

berr_en   enables bus errors for the entry.

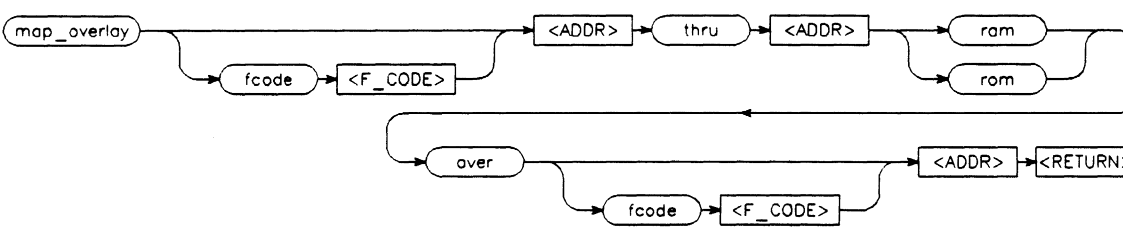
The first <ADDR> of a range specification should be the starting address of a block boundary. If an address inside a memory block area is entered, the system converts this address to the starting address of the block prior to its mapping. Leading zeros may be deleted if the most significant digit is 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 can't overlay map terms in the target system.

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 consists of 4K byte blocks of memory as figure 4-2 shows. The memory mapper hardware has a resolution of 256 bytes, the minimum map entry size.

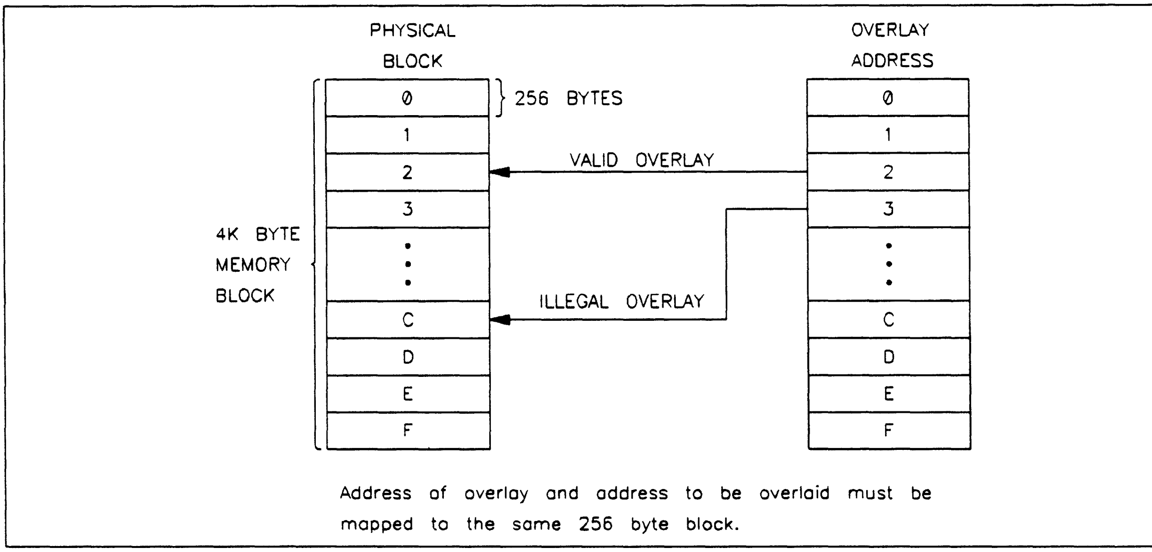


Figure 4-2. Overlay Addressing Within Physical Blocks

When specifying a memory address, the two least significant digits in a hexadecimal address (see figure 4-3) specify the address within the 256 byte entry. The third least significant digit specifies one of 16 256-byte entries within the 4K byte physical memory block.

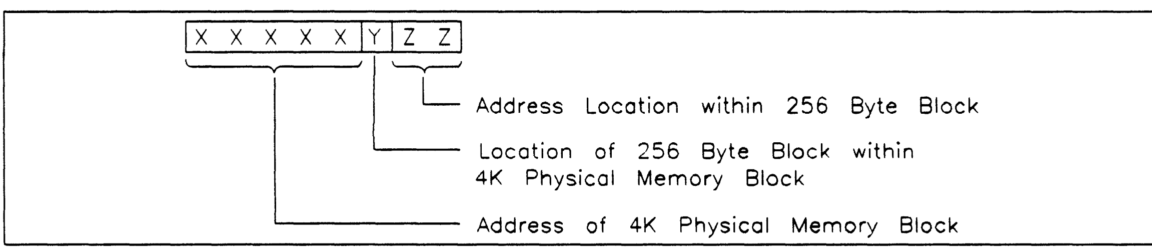


Figure 4-3. Hexadecimal Address Bit Definition

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. That is, 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 is valid. But, the command:

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

is not valid. 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 for a target system with some memory installed. This example shows how to use the **map** and **map_overlay** commands. Before you define the new memory map, delete all entries in the current map. Enter the commands:

delete all
modify defined_codes all

The memory map display will look like figure 4-4. Notice that one entry is still displayed. You cannot delete the CPU_SPACE

```
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 4-4. Sample Overlay Mapping #1

mapping to target RAM. This address space is required for vectored exception processing.

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

Add the following entries to the memory map:

> ***map fcode USER_DATA*** 0 ***thru*** 0ffffh ***emulation ram asynchronous width32***
>
> ***map fcode USER_PROG*** 18000000h ***thru*** 1800ffffh ***emulation rom asynchronous width32***
>
> ***map fcode SUPER_DATA*** 0 ***thru*** 3ffh ***target rom burst cache_dis berr_en***
>
> ***map fcode SUPER_PROG*** 0 ***thru*** 3ffh ***target rom burst cache_dis berr_en***
>
> ***map fcode SUPER_PROG*** 0f000000h ***thru*** 0f000fffh ***emulation ram asynchronous width32***

```
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  FFFFFFFFH   (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 4-5. Sample Overlay Mapping #2

```
map_overlay fcode SUPER_DATA 0f000000h thru
0f000fffh ram over fcode SUPER_PROG
0f000000h
```

The memory map resulting from these commands looks like figure 4-5. This is a typical 68030 memory map.

The map entries correspond to the following address spaces:

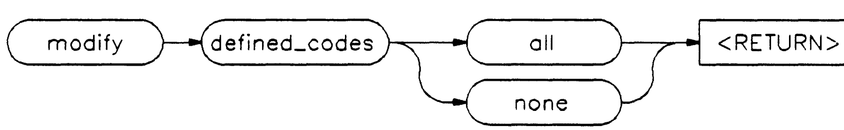
1. User application data space
2. User application program space
3. Exception vector table space
4. Emulation monitor data space
5. Exception vector table space
6. Emulation monitor program space
7. CPU_SPACE

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)" shows 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.

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.

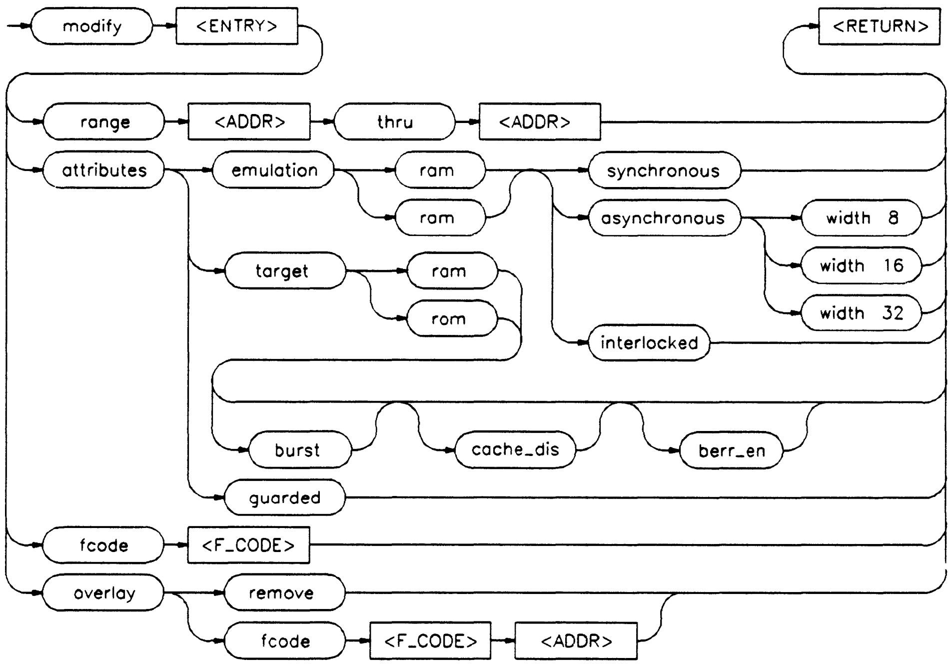
Modify Defined_Codes. The **modify defined_codes** command lets you selectively enable or disable the 68030 function code signals. The following diagram shows the command syntax:



where:

all	enables the memory mapper to use all three function code lines (FC0 through FC2) in mapping memory. If you select **all**, 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 you select **none**, 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:



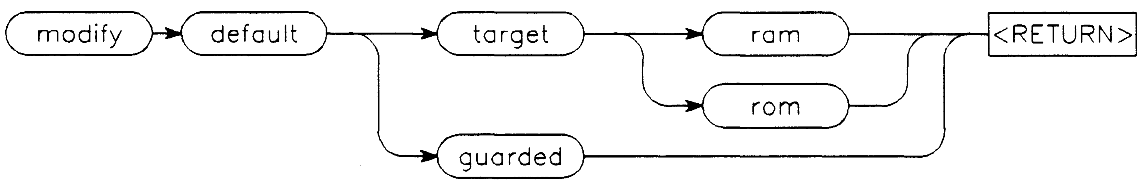
where:

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 available selections depend on the definition of the **defined_codes** parameter.

overlay — lets you remove an overlay from an entry. The entry is converted to the physical address corresponding to address specified in the entry. Or, you can change the function code or address range of the address space being overlaid.

Modify Default. Any address ranges that are not mapped when you end the mapping session are assigned the memory attribute specified as the default. Initially, the system assigns all unmapped memory to guarded memory. The default attribute can be set to target RAM, target ROM, or guarded by using the modify default command. The following diagram shows the command syntax:



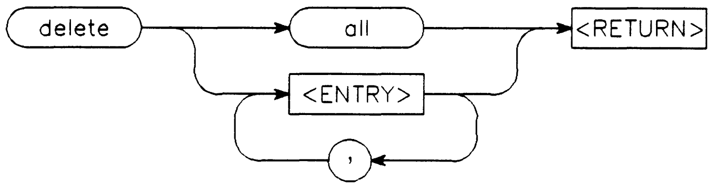
where:

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 that is not expected to be accessed. Any processor access to a location within such a range will cause a break to the monitor.

Deleting Memory Map Entries

Any memory map entry can be removed by using the **delete** command (except 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 look like figure 4-6. The deMMUer is described in chapter 5 and 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 4-6. Modify DeMMUer Configuration Display

Ending The Mapping Session

Exit the memory map configuration session by pressing the **end** softkey followed by **Return**.

Modifying The Emulation Pod Configuration

The following question asks you whether 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 emulator 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:

In-circuit emulation session?	no
Disable on-chip cache	yes
MMU enabled during session	no

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

Configuring for In-circuit Emulation Session

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? no (yes)
```

no — The emulator is configured out-of-circuit. The internal 20 MHz clock is selected. 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. The target system must provide a clock.

Enabling DMA Transfers

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)
```

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 passed 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, and so on) and meet the 68030 timing specifications.

CPU Clock Rate Determination of Wait States

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

no — If the clock rate is less than or equal to 25.0 MHz, all emulation memory accesses will occur with four wait states.

yes — If the clock rate is greater than 25.0 MHz, six wait states will be inserted for emulation memory and interlocked emulation memory accesses.

Disabling On-chip Cache

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

no — If the cache is left 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). A **no** answer improves system performance but much analysis capability is lost.

yes — The processor caches are disabled. You must answer yes to this question to use all analysis features.

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. See chapter 6, "Target System Interface," for more information regarding the on-chip cache.

Enabling MMU For Use During Emulation Session

```
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 must enable the MMU hardware and initialize the translation tables, root pointers, and so on.

Modifying Simulated I/O Configuration

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

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

Answering **yes** to this question prompts a series of simulated I/O questions. To learn how to answer these questions, see chapter 9. For more information about simulated I/O, see the *Simulated I/O Reference Manual.*

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

Modifying Simulated Interrupt Configuration

You enable simulated interrupts 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. You use simulated interrupts while the emulator is out-of-circuit to test software that depends on the occurrence of preemptive interrupts. Chapter 9 tells how to configure your system for simulated interrupts.

Naming The Configuration File

This question lets you name a file containing the emulation configuration information you have just entered. The configuration file is stored on disc and can be recalled 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. The information will be saved as the new default emulator configuration. To restore the original default file, 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. (See the *HP 64000-UX User's Guide* for more information on 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 means 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.
- Tells 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.
- Tells how to turn the deMMUer on and off.
- Describes the deMMUer configuration setup.
- Tells 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 you have the deMMUer board for the internal analyzer. If you are not using the 68030 MMU active mode, no address translations occur, and you can ignore this information.

Note

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

What The DeMMUer Is

The DeMMUer has hardware and software that improves 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 would be displayed (no symbols or source references). The deMMUer tracks MMU table walks to get the latest logical-to-physical translation information. Thus, the deMMUer can effectively translate the physical addresses to logical addresses. Then the analysis display software can lookup symbols for the addresses, and do 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 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 MMU.

Whenever 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 translation 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 command 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 a physical address. The analyzer will show this address in its trace list, but it cannot show any symbol associated with this address. Nor can it 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 these features during emulation:

- You want the trace list to show the assembly language activity captured during a trace. The inverse assembler needs sequential logical addresses to find the next piece of program information. Physical addresses probably will be non-sequential when crossing a page boundary.
- You want to use a trace specification that will be satisfied when a certain source file event occurs. To do this, you enter 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 while running a program.
- 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 because a symbol or source line resides at a particular logical address. The language tools establish that relationship. The source referencing knows nothing about physical addresses.

- 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 an operating system.

Unable to Do Reverse-Address Translations

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

- If the root pointers use **page** descriptor DT fields. Here, no table searches will occur. Physical addresses will equal logical addresses plus the offset given in the root pointer.

- 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 not work because it has resources for only one root pointer definition. The *68030 Internal Analyzer User's Guide* describes how to select a root pointer type. See that section for suggestions on handling this problem.

When To Start The DeMMUer

You can start the deMMUer and the 68030 MMU simultaneously, or you can turn on the deMMUer after the MMU has been operating. The following paragraphs discuss each case:

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.

Used Emulator without DeMMUer, Want To Use It Now

If you configured and enabled the deMMUer before 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 *68030 Internal Analyzer User's Guide* explains how to set up the deMMUer configuration display and enable the deMMUer. You will always have logical addresses when the 68030 MMU is off.

Turn On/Off By Using Configuration Questions

Invoke the emulation configuration questions by using the **modify configuration** command. Proceed through the questions until the following configuration question is presented, 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

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

To turn off the deMMUer, enter the command:

disable_deMMUer

Then 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

You can turn on the deMMUer from within an emulation session. You must have enabled the 68030 MMU, and have a valid value in the Translation Control (TC) register of the deMMUer configuration. You also must enable the deMMUer in the configuration. Enter the command:

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

Figure 5-1 shows the deMMUer default configuration display. 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 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 is presented:

```
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 can do 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 you use the emulator with a target system (in-circuit emulation):

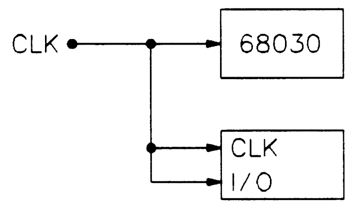
- Emulation and target system $\overline{\text{DSACK}}$ and $\overline{\text{STERM}}$ signals
- Vector base register
- The internal 68030 caches
- Using function codes for displaying and modifying reserved address space
- Enabling/disabling the bus error signal ($\overline{\text{BERR}}$)
- Using DMA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

Read this chapter before you try to use the emulator with your target system.

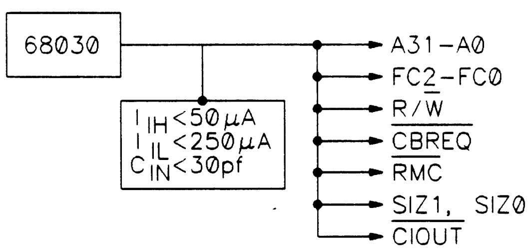
68030 Signals

The following section discusses each 68030 signal and how the pod interacts with each one. For a summary of the timing, AC, and DC specifications of the 68030 emulator see appendix C. This section shows the emulator/target electrical interface for each signal. The interface diagram is either with the signal definition or is referenced to a signal that has an identical interface. All interface circuitry 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 presents a greater load to 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, which reduces the amount of capacitance they can drive.



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

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

$\overline{\text{RMC}}$

The Read-Modify-Write Cycle line is not buffered to the target system. The emulator loads this signal, which reduces the amount of capacitance it can drive. 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 of capacitance they can drive is reduced. The emulator/target electrical interface is the same as shown for the address bus.

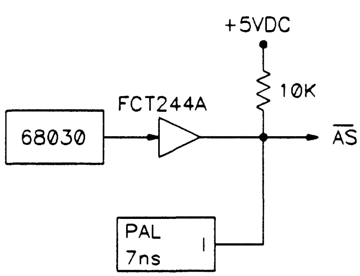
$\overline{\text{CIOUT}}$

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

$\overline{\text{AS}}$

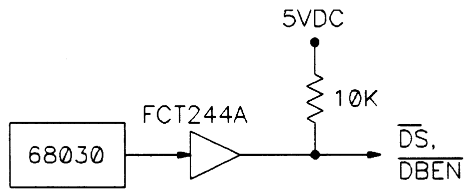
The 68030 Address Strobe signal line is buffered by the emulator at the target interface. $\overline{\text{AS}}$ is driven to the target when the processor is running, unless the emulator is in the background monitor, or the bus has been relinquished. The $\overline{\text{AS}}$ signal from the target system is treated as an input during DMA so that emulation memory and the analyzer can see those cycles.

Buffering the $\overline{\text{AS}}$ signal causes a signal delay. This delay may be significant in some systems, but in a system that has $\overline{\text{AS}}$ heavily loaded, it may not even be noticeable.



$\overline{\text{DS}}$, $\overline{\text{DBEN}}$

The 68030 Data Strobe and Data Buffer Enable signal lines are buffered by the emulator at the target interface. $\overline{\text{DS}}$ and $\overline{\text{DBEN}}$ are driven to the target when the processor is running unless the emulator is in background monitor, or the bus has been relinquished. Buffering these signals delays them slightly.

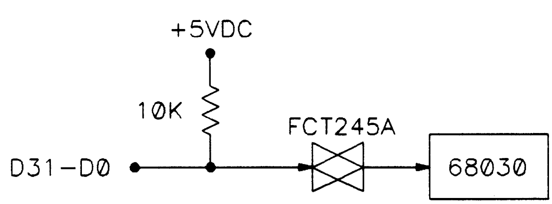


$\overline{\text{ECS}}$, $\overline{\text{OCS}}$

The 68030 External Cycle Start and Operand Cycle Start signal lines are buffered by the emulator prior to going to the target interface. $\overline{\text{ECS}}$ and $\overline{\text{OCS}}$ are driven to the target when the processor is running unless the emulator is in background monitor (when they are optionally driven), or when the bus has been relinquished. Buffering these causes a signal delay. The emulator/target electrical interface for these signals is the same as shown for the $\overline{\text{DS}}$ signal.

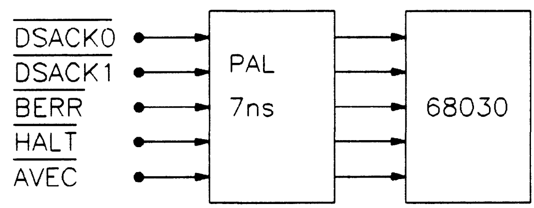
D(31-0)

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. The emulator requires more setup time than the processor because the data bus lines are buffered.



DSACK1-DSACK0

The Data Transfer and Size Acknowledge signals are buffered between the target system and the 68030. The target system signal 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 emulation memory has valid data. The emulator requires more setup time than the processor because the DSACK lines are buffered.



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.

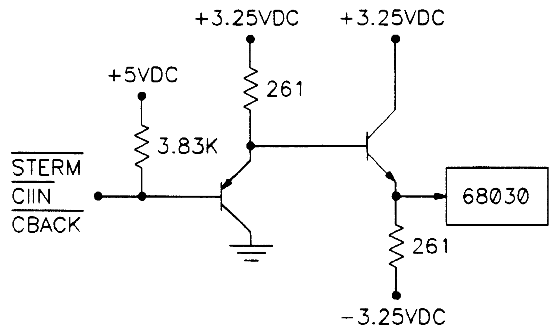
If HALT and BERR are asserted simultaneously, the BERR signal is not treated as a bus error; but as a retry. Retry cycles are never blocked by the emulator. The emulator/target electrical interface is the same as shown for the DSACK signals.

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

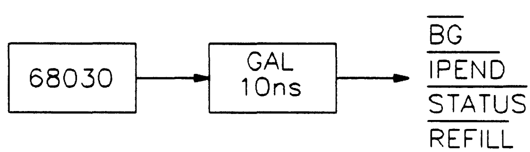
The Synchronous Termination signal is buffered between the target system and the 68030. The target system signal is only sent to the processor during normal cycles mapped to target memory, during interlocked emulation memory cycles, or during interlocked foreground data space cycles. During interlocked emulation memory cycles, the target STERM signal is not allowed until emulation memory has valid data. The emulator needs more setup time than the processor because the $\overline{\text{STERM}}$ signal line is buffered.



$\overline{\text{CIIN}}$ The Cache Inhibit In signal is buffered between the target system and the processor. The emulator doesn't block it. The emulator/target electrical interface for these signals is the same as shown for the $\overline{\text{STERM}}$ signal.

$\overline{\text{CBACK}}$ 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 $\overline{\text{STERM}}$ signal.

$\overline{\text{BG}}$ The Bus Grant signal is buffered between the processor and the target system. The signal is blocked when DMA is disabled.



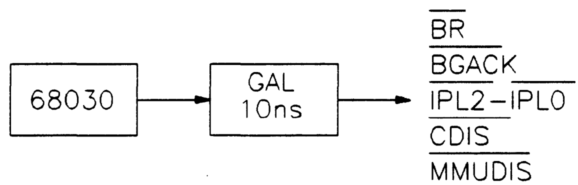
$\overline{\text{IPEND}}$ The Interrupt Pending signal is buffered between the processor and the target system. The signal is blocked during interrupts caused by the emulator break facility. You can configure the signal blocking. See chapter 4 for configuration information.

STATUS, REFILL

The emulator doesn't block the Microsequencer Status and Pipe Refill signals. 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

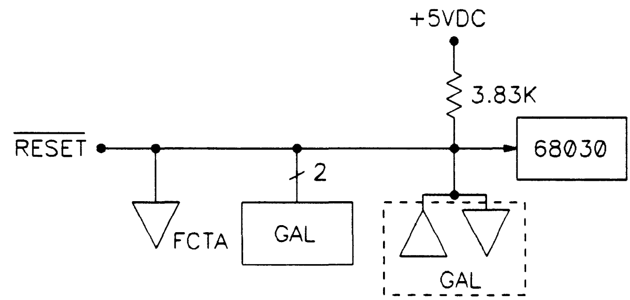
The Interrupt Priority Level signals are buffered between the target system and the processor. These signals are blocked while the emulator is in the background monitor. The emulator foreground monitor can delay interrupt handling. This is because the emulation monitor is itself an interrupt handler, and maskable interrupts are normally disabled during execution of the monitor. Maskable interrupts can be enabled during execution of some parts of the emulation monitor by customizing the emulation monitor code for the specific application. For further information on the emulation monitor and how to customize the code refer to chapter 7. The emulator/target electrical interface for these signals is the same as shown for the BR signal.

CDIS, MMUDIS

The Cache Disable and MMU Disable signals can be blocked by the emulator. Whether these signals are blocked is determined by the emulator configuration (see chapter 4 for more information). The emulator/target electrical interface for these signals is the same as shown for the BR signal.

RESET

The Reset signal is not buffered by the emulator, although the emulator can drive this signal because it is an open collector signal.



VCC

The emulator monitors the target system VCC to determine when the emulation pod is connected to an active system. The target system interface is disabled until VCC is detected. The current draw from the target VCC is a few milliamps.

Emulation And Target System DSACK and STERM Signals

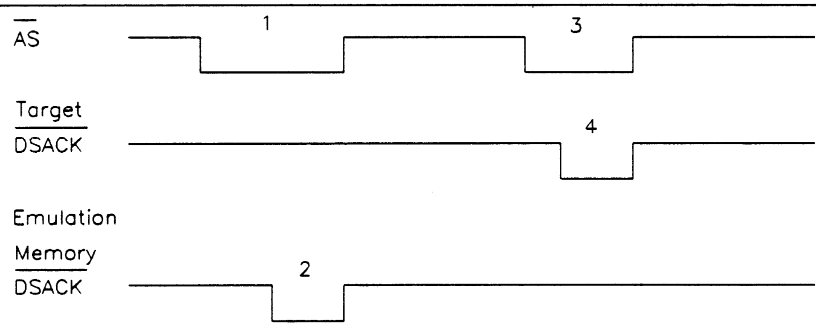
Interlocking Emulation Memory and Target DSACK and STERM Signals

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. Then accesses to emulation and target memory will show system performance similar to that of the processor only.

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 you don't interlock emulation and target DSACK and STERM signals. See the timing diagram in figure 6-1.



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

Use the following rules to decide whether to interlock emulation and target DSACK and STERM signals.

- If the target system generates DSACK and STERM signals for all emulation memory address ranges, interlock the emulation and target DSACK and STERM signals.
- If the target system does not generate DSACK and

STERM signals for a range of emulation memory, do not interlock the emulation and target DSACK and STERM signals.

- If there is no target system (out-of-circuit emulation), you cannot interlock DSACK and STERM signals.

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 for the emulator. They usually cause improper data fetches from memory and cause target system failure with the emulator installed.

Use Of Open Collector Drivers

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

Improper values for pullup resistors can cause slow signal pullup. The signals may interfere with the next cycle. The pullup resistor value is too large to return DSACK and STERM to a proper high level before the next cycle begins. The still low DSACK and STERM signals terminate the second cycle prematurely, causing improper data fetches by the CPU.

Early Removal Of DSACK Signals

Some target system designs do not follow the 68030 specification that 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 causes "no DSACK" messages in the trace list and inverse assembly failure. The emulator may fail completely 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. Trace 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 at the address contained in the Vector Base Register (VBR).

The 68030 emulator uses a jamming technique for breaks and software breakpoints. Therefore, most monitor functions don't need the value of the VBR. The vector table may be located anywhere without adversely affecting emulator operation.

When you use the foreground monitor, the single-step feature does need the trace exception vector (VBR + 24H). If you use this feature, 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, then waiting for user intervention. These exceptions include Bus Error, Address Error, Divide by zero, and so on. If you use these exceptions, you must maintain the exception vector table so that the vectors always point to the appropriate monitor location.

Using the Internal 68030 Caches

Using the internal 68030 caches affects several emulator functions.

Cache Control

When the emulator is operating out-of-circuit, the "Disable on-chip cache?" configuration question has a different interpretation than when plugged into a target system.

- When out-of-circuit, a "no" answer to the "Disable on-chip cache?" configuration question forces the $\overline{\text{CDIS}}$ signal high within the pod.

- When in-circuit, a "no" 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 "yes" answer forces $\overline{\text{CDIS}}$ low within the emulator.

If the target system uses the internal 68030 caches, the caches must be enabled by answering "no" to the "Disable on-chip cache?" configuration question.

Recall that the target system $\overline{\text{CDIS}}$ must be high, and bit zero of the Cache Control Register (CACR) must be set to 1 to enable the instruction cache. Bit 8 of the CACR must be set to 1 to enable the data cache. You can set bits 0 and 8 of the CACR as follows:

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

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

The caches are not frozen on entry to the foreground monitor. The cache contents are overwritten.

If you set a breakpoint for an address currently contained in cache, the breakpoint isn't recognized until the CPU fetches from that address in main memory again. The **run until** command is similarly affected, because the command implementation uses breakpoints.

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 true with tight program loops and with high performance code segments that are frequently locked in cache. Since the analyzer cannot capture internal cycles, it cannot display these cycles in the trace list. This can result in missing trace data and high-level source lines, and even improper disassembly. The analyzer also will 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 is timing loops). Sometimes you can select trigger and store patterns that correspond to external cycles. If there are normally no external cycles, try to place "markers" in the cached code so that the code will generate an external cycle for analysis purposes.

You can disable the cache for pages of target memory when mapping memory. This improves the analysis trace list.

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

Using Breakpoints With Caches Enabled

Sometimes, 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 code segment (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, because the CPU does not generate an external program read cycle to memory and therefore never "sees" the breakpoint.

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 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. The host can set breakpoints in dual-port emulation memory without using the emulation monitor. Thus, setting a breakpoint won't clear the 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 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 also apply to disabling software breakpoints.

Function Codes For Reserved Address Space

When you enable function codes during a memory mapping session, the **display** and **modify** commands use the function codes specified in the command. When function codes are disabled, all memory references in commands assume function code 0.

Some target systems do not use function codes to differentiate between user and supervisor space or program and data space. They do decode the "reserved" address spaces (function codes 0, 3 and 4) to generate interrupts or inhibit DSACK generators. You can customize the emulation monitor to allow the use of a nonzero function code for the **display**, **modify**, **load**, and **store** emulator commands.

To modify the monitor, change 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 nonzero 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. Arrows (>>>) show the added and modified lines.

Enabling/Disabling BERR

The 68030 emulator allows the bus error (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, set the emulator configuration to block BERR. This would normally occur if DSACK or STERM signals are not generated for emulation memory accesses.

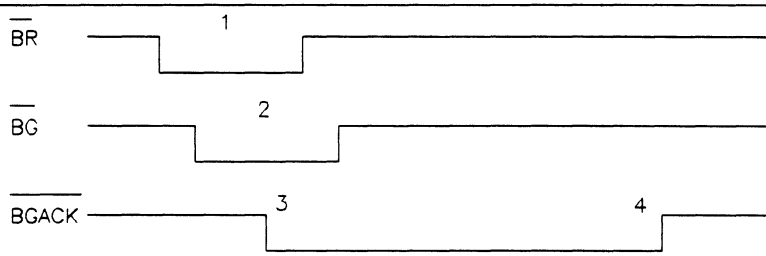
If the target system generates DSACK or STERM signals for emulation memory accesses, then it probably does not generate BERR for these cycles. Here, BERR indicates a failure, and should be enabled in the emulator.

Using DMA

If any devices share the 68030 bus and can perform DMA, then DMA should normally be enabled. This enables the CPU to receive the Bus Request (BR) signal, generate a Bus Grant (BG) response signal, and receive the Bus Grant Acknowledge (BGACK) response from the bus requester. Figure 6-2 shows the handshake sequence for DMA transfers.

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

If you have enabled DMA, you can enable or disable DMA to/from emulation memory.



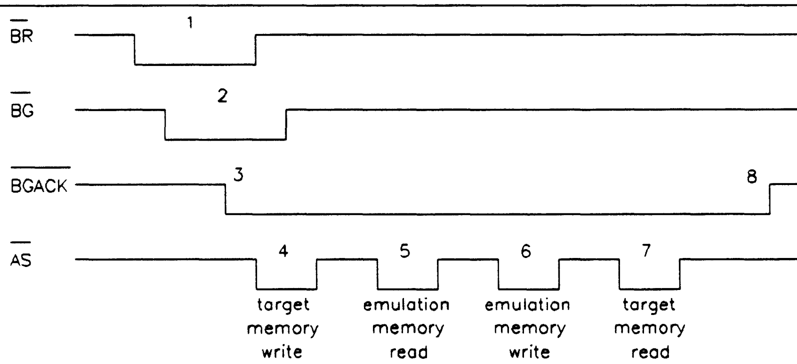
1. External device requests the bus.
2. The 68030 indicates that the bus will be granted.
3. External device indicates that the bus is in use.
4. External device relinquishes the bus.

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

If DMA to emulation memory is enabled, the DMA hardware can read from or write to emulation memory. If DSACK or STERM signals are interlocked, the target system supplies the DSACK or STERM signals for these accesses. 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 will hang the DMA hardware if DSACK or STERM signals are required for cycle termination.

If DMA to emulation memory has been disabled, the DMA cycle is permitted, 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

You must use the **run** command properly to avoid serious, stack related problems in the target system software.

A primary cause of target system "failure" is the incorrect setup or restoration of the stack when you use the **run** command. A 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 called the "calling sequence.") Suppose that a procedure PROC1 expects the stack frame shown in figure 6-4.

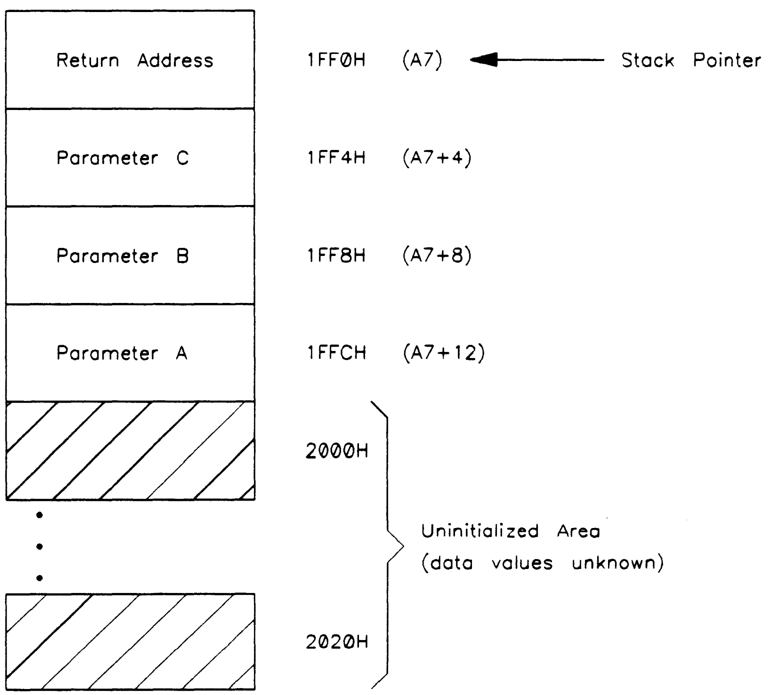


Figure 6-4. Example Stack Frame

Often, PROC1 will access the stacked parameters by referencing parameter requests to the stack pointer. This means that parameter "A" is 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 cause 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 fall within guarded memory area, causing 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** also can cause problems. This 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 the user program from the point where it stopped. 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** command. Chapter 10 explains how the **run** command works.

Using the Foreground Monitor

Loading the Monitor

Follow these rules when you load the emulation monitor:

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

  In addition, parts of the monitor must write to other monitor program locations. Since writes to ROM are always blocked, the program and data sections of the monitor must be mapped to RAM.

- 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 monitor, the symbol addresses are assumed to be associated with the SUPR_PROG function code (since the monitor is 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 uses 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 you're using the background monitor, the initial stack pointer must be defined since stacking is done in the 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. SP_TEMP is provided only as a small temporary stack for monitor use. So the stack may overflow easily once a **run from ...** command is given, and the target system program begins execution. Parts of the monitor may be overwritten if the SP_TEMP stack overflows.

To ensure proper operation, 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. Do this by including an "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 solution is to be certain that software execution started by the **run from ...** command 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 emulator caused the present reset condition. The presence of a lower case "reset" means that the target system is presently asserting the CPU reset signal. You can have the 68030 emulator enter the emulation monitor when a **run** command is issued after "Reset." (Jamming occurs only if the reset signal is asserted by the emulator.) The emulator ignores the initial program counter and initial stack pointer vector. Instead, the jamming circuitry supplies these values based on the current location of the monitor.

You may have problems if the target system does some hardware and/or software initializations on reset. If "reset into monitor" is used, these initializations are not done before monitor execution is begun.

Memory Access Timing Issues

Access time is the 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,$\overline{\text{RMC}}$,$\overline{\text{CIOUT}}$,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 +14ns -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)

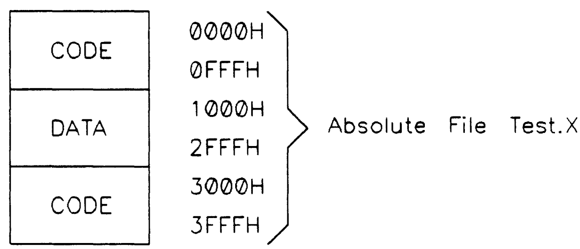
- Spec. 27 = 6 ns (min)
- Cycle Time = 30 ns (min)
- Clock Pulse Width = 14 ns (min)

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 often has various "sections" containing code or data:



A memory map resembling that shown below might normally be generated:

Address 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 because the **load** mechanism attempts to load the entire file using the SUPR_PROG function code. In 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 mapped to GUARDED memory when the function code is SUPR_PROG. This is because the default is set to GUARDED, and 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 loads all memory areas present in a given absolute file. (Guarded, as well as target and emulation memory.)

The **load memory emulation** . . . command loads only those areas mapped to emulation memory in a particular absolute file.

Thus, to properly load Test.X, you would use the following three commands:

```
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 loads only those areas mapped to target memory in a given absolute file.

Debugging Plug-in Problems

When you connect the emulator to a target system, the emulator operation becomes more complex. More hardware has been added to the system. You must be knowledgeable about the target system resources. This section is a guide to isolating problems that you encounter 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 or the target system timing requirements.

Review the Configuration

An incorrect configuration file can cause improper operation. Review the configuration file to ensure that all questions are answered correctly for your target system. If you are not sure how to answer a question, see 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 can operate without the emulation pod usually can start with the default configuration file. Use this file whenever you start a new emulation session. The default configuration enables all target system signals, maps all memory as target RAM and does not load the emulation monitor. Make sure that the target system operates correctly by using the internal analyzer and indications from the target system. Isolate plug-in failures with the default configuration 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 emulation. It passively monitors each processor bus cycle. Analyzer data can be displayed without disrupting the emulation process. You can use the analyzer to verify the proper program and target system hardware operation.

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 analyzer's triggering capability can be used to capture conditions that are caused by a failed interface. Use 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 specific operations such as a CPU space address or a particular place in memory can be debugged using the analyzer's capability to drive a rear panel BNC output 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 monitor the target system. When observing the data, remember 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 lists the emulation status line messages and their causes. Many conditions are not displayed unless no bus cycles have occurred for more than 250 milliseconds. If your system normally 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.

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 . . .

Sometimes the system will malfunction though the emulator is configured correctly and the target and monitor programs are loaded. 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, and so on, will crash if the delta time interrupt is not serviced within a preset time limit. You can customize the foreground monitor to enable or disable interrupts as required. See the "Continuing Target System Interrupts While In The Emulation Monitor" section of chapter 7.

You can disable the normal target system function of the level 7 (NMI) interrupt through vector table modifications, and some additional foreground monitor code.

Ensure that the target program is not accidentally overwriting the foreground monitor or vice versa. You can use the analyzer to examine software behavior. This is an effective way to solve emulation problems. Obtain a listing of the foreground monitor and the target program, 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. Then you can examine CPU activity surrounding the foreground monitor entry. Observe the stacking activity of the level 7 interrupt, as well as emulator generated jam cycles. This will help you decide whether 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 target programs, and may cause hardware and software anomalies.

The Emulation Monitor Programs

Overview

This chapter:

- Explains why you need an emulation monitor program.
- Compares the foreground and background emulation monitor programs.
- Suggests when to use either a foreground and background monitor.
- Explains how the break function is related to the emulation monitor.
- Describes the foreground emulation monitor program.
- Tells how to customize the foreground monitor.
- Lists the foreground monitor memory requirements.
- Describes the foreground monitor linking requirements.
- Lists the rules for loading the foreground monitor.

See chapter 6, "Target System Interface," and chapter 10, "How the Emulator Works," for more information about the emulation monitor and its interactions with the host computer and your target system.

Introduction

The emulation monitor program implements many emulator functions. These 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.

Comparison of Foreground and Background Monitors

An emulation monitor satisfies 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. The emulator system controller reads these without further interference.

Background Monitors

A *background* monitor is an emulation monitor that overlays the processor's memory space with a separate memory region. Entry into the monitor is done 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 on powerup. You don't have to worry about linking 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 you write any target system code. All processor address space is available for target system use, because the monitor memory is overlaid on processor memory, not subtracted from it.

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 needed 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. The foreground monitor does use part of the processor's address space, which might cause problems in some target systems. You also must configure the emulator to use a foreground monitor. (See chapter 4, "Answering Emulation Configuration Questions.")

You may link the foreground monitor with your code. Linking the monitor separately is preferred. Then you can download the monitor before the rest of your program. Linking monitor programs separately is more work initially. It can improve efficiency later, because you can load it separately during the configuration process at the beginning of a session.

Choose a Foreground or Background Monitor

Most conventional emulators use either a background or foreground monitor as the emulation control program. The emulator designer makes the appropriate compromises regarding the emulator's transparency and chooses one type of monitor or another in implementing the emulator.

Because the emulator supports an on-chip MMU and has full virtual system support, it supports both background and foreground monitors. You choose a monitor based on the development stage and nature of the target system.

When to Use the Background Monitor

You should usually use the background monitor during the early stage of hardware development where full functionality of the target's interrupt, bus error, and other asynchronous events is not yet needed. The background monitor has the advantage of being easy to use. It can enable the emulator to be a stimulus for help in turning on the target hardware without requiring full target system functionality. For example, the display/modify target memory feature can be used to stimulate the target's memory interface and help you troubleshoot any defects in that circuitry. All the emulator would show while in background is the bus cycle referencing the target address. By using an external timing analyzer (for example the HP 16500A), you can monitor the target's signal behavior during that cycle and find any problem(s).

Another feature of the background monitor (that is not in the foreground monitor) is the display/modify physical memory. The 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 completion, 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. You can change the monitor source code to fit the particular application. Messages relating to certain events can be added and displayed on the emulation terminal, and target programs can jump to the monitor. You can display or modify coprocessor registers by adding the proper code to the monitor program.

In systems that use the 68030's on-chip MMU, external memory in the target system and emulation memory are accessed as physical addresses. Because 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 foreground monitor does not need to be mapped to emulation memory. Additional emulator hardware allows linking the foreground monitor 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 in memory mapped to the untranslated CPU space of the processor. This makes it easier to install and use the foreground monitor.

Customizing the Monitor Programs

You can't customize the background monitor.

The source code for the foreground monitor comes with the HP 64430 and can be modified to best fit the target application. Another section of this chapter tells you how to customize the monitor.

The Break Function and the Emulation Monitor

The emulation break circuitry uses the NMI (INT7) resource of the processor to interrupt the user program and enter the emulation monitor. 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 covers both foreground and background monitors, but you can't access symbols or entry points in the background monitor.

The emulation monitor has 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.

The following paragraphs discuss each section.

The Exception Vector Table

The emulation foreground monitor is entered through processor exceptions. The monitor contains pseudo instructions that load the vector table with the addresses of the monitor exception handlers. The monitor exception table predefines some 68030 exception vectors for your convenience.

The emulation monitor program has all exception vectors (except RESET and MONITOR SINGLE STEP) contained in comment fields. This allows you to supply the addresses for custom 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 table that comes with the monitor program.

If your application has a RESET handler, modify the reset vector in the monitor to point to the user reset handler. Also, you must disable the reset-to-monitor function. Do this by modifying the emulation configuration. You must answer **no** to the configuration question "Reset into the monitor?." See chapter 4 for details.

Note

The monitor's 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. There are six separate paths for monitor entry. Each path is distinguished from the others by a unique ENTRY_ID code, which is stored on 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 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. The monitor saves the processor's registers and restores the interrupt mask (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 (that is, a BKPT instruction inserted in your code by the HP 64000-UX system) occurs.

JSR_ENTRY

Use the JSR_ENTRY (foreground monitor only) entry point if you want your target program to jump directly into 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, use a trap exception. The trap vector should point to MONITOR_ENTRY.

RESET_ENTRY

RESET_ENTRY is the entry point when the 68030 processes the reset exception. RESET_ENTRY sets up a default stack and initializes the processor's registers to default values.

EXCEPTION_ENTRY

A set of exception entry points (foreground monitor only) give status messages for the ten exception vectors after reset. These exception vectors are for your convenience and may be deleted or modified. For more information, see the foreground monitor source program and the section in this chapter titled "Modifying The Exception Vector Table."

Emulation Command Scanner

The emulation command scanner normally rests in an idle loop labeled MONITOR_LOOP. The host repeatedly examines the system global MONITOR_CONTROL. If bit 15 is zero, the idle loop is resumed. If bit 15 is one, there is a command, and the program branches to the appropriate command routine.

The host sets bit 15 of MONITOR_CONTROL, the monitor program clears it. The lower byte of MONITOR_CONTROL contains a command number. The command table is searched for this number. If there is a match, a command entry point is retrieved from the table and the command will be executed. Otherwise, the program will return to the idle loop. The command is 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

The host (the HP 64000-UX system) uses ARE_YOU_THERE to determine whether the processor is executing in the monitor or in the target system code. It also can 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 delays EXIT. The monitor loops until it receives an emulator status bit that indicates a synchronized start among multiple emulators. Then it executes EXIT. Any command will abort the wait loop.

COPY_MEMORY

COPY_MEMORY moves data between the monitor parameter block areas and target system memory. This routine 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 that contains the address offset of the coprocessor register load/unload routine for each of the eight 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 that allows you to implement interrupt driven code on an emulator that is out of circuit. This function must set the local simulated interrupt enable flag TRUE and store SIM_INTS_TRUE at SIM_INT_CONTENTS to reenable 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 that allows you to disable interrupt driven code on an emulator that 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 that allows you to implement interrupt driven code on an emulator that is out-of-circuit. Usually, you'll have this routine 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.

Using and Modifying the Foreground Monitor

A standard foreground emulation monitor source file comes with each emulation system. You must assemble and link this file before using it. Typically, you'll assemble the monitor and link it with the user program to form one software module. Then you load this module into memory.

You can modify the foreground monitor to suit a particular target system or to expand the monitor's capabilities. Some foreground monitor functions won't work until you remove the comment delimiters from the code for those functions. If you modify the foreground monitor, you must maintain the communication protocol between the monitor and the emulation software.

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 destabilize the system and make it unusable.

You should not change any parts of the monitor other than those described in the following paragraphs. Changes in other sections may cause some features to stop working due to stack modifications, or because the information that is passed to and from the various sections has been affected.

For most systems, the foreground monitor supplied with your emulator enables all emulation features to operate. Some systems need a custom monitor program to maximize the emulator's effectiveness. So, the emulator comes with a monitor source program, which is thoroughly commented. The comments describe each standard routine so that you can easily make your modifications.

Caution

Possible loss of original monitor source program! Do not modify the original monitor source program. You should copy the 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 with the 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 originally has "read-only" permissions.

You should now modify the copy in your subdirectory.

Note

After you modify the monitor, be sure to reassemble and relink it.

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 use your editor to 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 your editing 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

You can restore the processor interrupt mask to its pre-break value to enable target system interrupts while in the monitor. You must edit the monitor program if you want to enable interrupts while running in the monitor.

Under the MONITOR_ENTRY label, you will find a commented section that describes reenabling 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 User Program Messages to the Display

Note

This option is available only with the foreground monitor.

The PUT_MONITOR_MSG routine in the emulation monitor provides a way to send messages to the display status line. To use this feature, you must:

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 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.

Below is an example program implementing the "message breakpoint."

```
*********************************************************************************
*
*    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
*******************************************************************************
```

Monitor Memory Requirements

The emulation system divides emulation memory into 256-byte blocks. 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 check the exact value by examining 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 you're using the standard emulation monitor. To check the memory requirements for the emulation monitor being used, check the linker listing file.

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

Follow these rules when you load the emulation monitor:

1. Data space of the monitor must be mapped as RAM as opposed to ROM. The monitor transfer buffer and many monitor "housekeeping" variables must be read and write accessible, and must, therefore be mapped as RAM.

   In addition, parts of the monitor must write to other monitor program locations. Since writes to ROM are always blocked, the program and data sections 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. For the monitor, the symbol addresses are assumed to be associated with the SUPR_PROG function code (since the monitor is 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:

- Discusses the requirements for using custom coprocessors.
- Describes the custom coprocessor format file.
- Tells how to modify the emulation monitor for use with custom coprocessors.
- Explains the emulation configuration questions related to custom coprocessors.

Introduction

Note

Only the foreground monitor supports custom register access (except for the MMU registers). Both the foreground and background monitors support display and modification of MMU registers.

The 68030 emulator can access floating point coprocessors and other coprocessors in your target system. You can both display and modify coprocessor register sets.

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 comes with your emulation software. This file is named:

```
/usr/hp64000/inst/emul32/0410/0204/
custom_spec
```

Read/write routines for the MMU are 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.

- The coprocessor space in which the coprocessors are located.

- Size of the register buffer for data transfers.

- Display format for each coprocessor.

- What register names are available for register modifies.

This file is read when the emulation configuration file is processed. 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. The quotes and brackets do not correspond to search paths. Instead the filename must be the full pathname for the include file.

Include files simplify your custom register specification file and allow you to remove a register set from the specification file.

Figure 8-1 (at the end of this section) lists a sample custom register specification file. Figures 8-2 and 8-3 show how the same file could be written using an 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 move 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 contain only 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

Enclose the register set display specification 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 will generate the register display. These lines also define register names for the **modify** command. Register specifications have the form:

```
NAME %OFFSET.WIDTH
```

where:

NAME	is the name used for the register in the **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.                              */


/*************************************/
/* 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 Specification 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  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 8-2. Custom Reg. Spec. File Using Include Files

```
/*************************************/
/* 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 8-3. Custom Reg. Spec. Include File fpu_spec

Emulation Monitor Changes

To access coprocessor register sets, you must change 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 to save register values read from or written to the custom coprocessor. An example buffer (MMU_REGS) is declared in the emulation monitor program.

```
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
```

Find this declaration in the emulation monitor program and insert your custom coprocessor register buffer declarations immediately after it. 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_REGISTERS 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 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 is as follows:

```
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. The emulation monitor gives an example read/write routine (FPU_881_COPY) for use with an external FPU. The default MON_ALT_REGISTERS table is as follows:

```
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.

The following listing shows an external FPU read/write routine (FPU_882_COPY). The external FPU copy routine is an example of how to write a 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
*       CMPI.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 Coprocessor Configuration Questions

After you modify the emulation monitor, you must assemble it and link it with your user file.

The final step in setting up 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.

Answer the remaining emulation configuration questions and save your changes to a configuration file. Now you can run emulation using custom coprocessors.

Chapter 4 gives a complete description of the emulation configuration questions.

Notes

Using Simulated I/O And Simulated Interrupts

Overview

This chapter:

- Tells you how to configure simulated I/O, with a section on simulated I/O restrictions.
- Discusses simulated interrupts, including:
  - How simulated interrupts function.
  - Simulated interrupts versus real interrupts.
  - Simulated interrupt configuration.
- Explains 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 you enable the MMU, 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 is in the *HP 64000-UX Simulated I/O Reference Manual.*

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

no — Answering **no** skips the simulated I/O questions. The current simulated I/O configuration is unchanged.

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 will 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 reenter control addresses or the file names for standard input, standard output, and standard error output. Answering no skips the remaining simulated I/O questions.

yes — The emulation software will frequently read the control address to see if the user program has requested any simulated I/O commands. Answering **yes** prompts the following questions:

`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 you load a file containing that symbol. 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 in emulation memory. Using target RAM causes a break to the monitor program when the control address is polled for simulated I/O commands or data.

The following questions assign 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 as shown.

These files are not opened until Open (90H) is called with the file names "stdin," "stdout," and "stderr." These files 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 reserved files automatically if simulated I/O is used. 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 time.
- There can be only four active simulated I/O processes at any time.
- When using the MMU, all simulated I/O control addresses must be mapped 1:1.
- When using the MMU, the memory for simulated I/O must be accessible in the supervisor state of the processor.

Since any open simulated I/O file 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 time.

Simulated Interrupts

Simulated interrupts allow out-of-circuit testing of software that depends on the occurrence of preemptive interrupts. You enable the simulated interrupt facility by writing the value 0ffh to the simulated interrupt control address. The control address is defined during emulation configuration. The simulated interrupt facility generates approximately six interrupts per second, depending on what other emulation activities are occurring concurrently (such as simulated I/O and display updates).

You can use simulated interrupts 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 to service the interrupting device. Normal execution resumes after the device has been serviced. Interrupts are asynchronous to normal program 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 flushes the current instruction counter, the processor can't continue program execution. 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. So the emulation monitor must be loaded 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 breaks to the monitor.

The monitor saves all registers during the monitor entry sequence. It then loops, waiting for a command. The emulation software then sends a simulated interrupt command to the monitor. The default monitor contains only a stub that immediately signals completion.

However, a simulated interrupt is user definable. To create a simulated interrupt, you must modify the emulation monitor. Include the interrupt code needed to perform the actions you want 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, see the section of this chapter titled "Modifying the Monitor to Use Simulated Interrupts."

After the interrupt is serviced, emulation sends the exit monitor command to the monitor. The exit monitor routine restores the registers that were saved on entry to the monitor, which continues normal program execution at the point where it was interrupted.

```
****************************************************************************
* 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 tha 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 occuring, 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    #0FFH,SIMINT_CA  ;Enable simulated interrupts.
                  MOVE.L    #0FFFFFFFH,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

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. The simulated interrupt configuration specifies the maximum time that emulation should wait for an interrupt handler to finish. If the interrupt handler does not complete within the specified time, emulation forces a break to the monitor, reports a failure, and terminates the simulated interrupt. It is not always possible to wait for simulated I/O to complete an interrupt handler.

The emulation software may appear to do several things concurrently:

- Polling up to six simulated I/O control addresses.
- Polling a simulated interrupt control address.
- Updating a display.

But, a single HP-UX task does each of these emulation tasks sequentially. This means that the simulated interrupt must complete before any other tasks can begin. This is why the simulated interrupt handler has limited execution time. If the handler is allowed to run indefinitely, the emulation program can be "locked up."

The final difference between simulated interrupts and real interrupts is that a simulated interrupt cannot 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 presented. When real-time mode is disabled, the command line displays the 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. These are:

```
Enable polling for simulated interrupts? no
(yes)
```

no — if you select **no**, emulation does not poll the control address and never causes a simulated interrupt.

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 for the simulated interrupt control address.

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, then reload the program without loading the monitor, the symbolic control address might be changed. The monitor program will not recognize the change unless you reload it.

If you do not reload the monitor when you load the target system program, you must ORG the control address to a specific location. If you ORG the address, modify the "**Simulated interrupt control address**" configuration question to point to the new address.

Also, you can link the monitor with your program. Then the monitor recognizes any new address because it loads with your program.

A similar situation occurs if you modify the control address configuration question. If you are running your program, 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>)

Your answer specifies the time, in milliseconds, to allow a simulated interrupt handler to execute before concluding that the handler has failed. (The emulator will then break to the monitor.)

The default time is 25 milliseconds. This time is approximately equal to the time required to initiate a simulated interrupt and check for its completion on an HP 9000. Though the resolution of this specification is one millisecond, the effective resolution is approximately 15 milliseconds. This is due to the time that is required to check for completion. For example, changing the maximum delay from 25 milliseconds to 26 milliseconds probably won't affect execution. Emulation does not always wait for the maximum delay. 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 must be in the range of 1 through 10000. Therefore, the maximum delay is 10 seconds. This upper limit prevents "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: "**ERROR: Simulated interrupt failed to complete.**"

Restrictions On Simulated Interrupts

Restrictions on the use of simulated interrupts are:

- The background monitor doesn't support simulated interrupts.

- When using the MMU, all simulated interrupt control addresses must be mapped 1:1.
- When using the MMU, memory for simulated interrupts must be accessible from the processor's supervisor state.

Modifying The Monitor To Use Simulated Interrupts

The user defined simulated interrupt function allows you to implement interrupt driven code on an emulator that is out of circuit. This command typically branches to your interrupt handler with a TRAP instruction. It must set the boolean variable SIM_INTS_ENABLED to TRUE and copy the control address to SIM_INT_CA. Then the monitor can disable simulated interrupts on entry. Otherwise, the **break** softkey will not function.

The monitor program must be modified before you can use the simulated interrupt feature. Find the block of code shown in figure 9-2 in the monitor program.

The TRAP #14 instruction will service the interrupt routine. You must uncomment the instruction. Or, if you want to use a different instruction, you must put it 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 finish editing the monitor, be sure to save your changes. You must reassemble and relink the monitor 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

Notes

10

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

This chapter will help you understand how the emulator works and how it interacts with your target system. This information and that in chapter 6, "Using the Emulator," can help you use the emulator more effectively and avoid problems that can occur when you use the emulator with a target system.

Note

The algorithms described apply to both background and foreground monitors unless otherwise specified.

Are You There Function?

The host computer uses the "are_you_there" monitor function to see whether the 68030 CPU is executing the monitor. It is used mostly to display the "running" and "running in monitor" status line messages.

It also makes sure 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 monitor. If the break fails, the host computer cannot complete the 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 clears bit 15 (**MONITOR_CONTROL** = 0), and returns 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 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 returns (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 returns (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. It sets the 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.

   a. If the command specifies "SUPERVISOR_STATE," the host sets the SUPERVISOR/USER bit to 1 (supervisor). Then 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). Then the 68030 CPU will execute in user mode on exit from the monitor.

7. The host returns (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 can be inserted into your program to help in debugging. The **run until** command also uses software breakpoints.

Note

When you use the foreground monitor, the exception vector table is referenced only for permanent breakpoints, which use the trace exception vector (VBR+24h). If one-shot breakpoints are working correctly, but permanent breakpoints fail, ensure that the trace exception vector properly references the monitor (memory location **MONITOR_ENTRY**).

Setting A Software Breakpoint

When you execute the command **modify software_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 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 reads 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 returns (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 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 make sure 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

The host computer determines if the last BKPT instruction executed was permanent or one shot.

1. If the breakpoint is one-shot, the emulation monitor returns* (RTE) to the user program and begins execution at address ADDR.

2. If the breakpoint is permanent and the "run from" address is 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.

3. 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 stack is modified 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 implementation of the single-step function 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 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>. Thus, on 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 that 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 tells 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 using 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>. On returning from the monitor to the user program, program execution will begin at <address>.
5. The host returns (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. You can tune this algorithm to meet specific target system requirements.

Alternately, the display/modify 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 aligns the target system address 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 below.

Starting Address Bits		Transfer Description
1	**0**	
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"

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 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 is transferred.

c. If a single byte remains, a byte is transferred. This monitor function is summarized below.

Number of Bytes Remaining	Transfer Description
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

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. Then it copies the memory locations to an internal monitor buffer for transfer to the host computer. This process is as follows:

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 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:

Description	Monitor Location
Number of bytes to read	PARM1
Starting address of target system read	PARM2
Function codes for target system read	PARM3
Starting address of monitor data buffer write	PARM4
Function codes for monitor data buffer write	PARM5
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**. The monitor exits the idle loop and begins execution at monitor program symbol **COPY**.

6. The monitor sets up the transfer according to the six parameters listed above. Then it copies target system memory values to the monitor data buffer using the algorithm described in the previous section. See the emulation monitor listing for 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 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 by 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 with 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 as follows:

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 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. Then it copies 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 supplied 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 by writing the "exit monitor" command (8001H) to **MONITOR_CONTROL**. This 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 with 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 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 see whether the monitor is executing.

4. The host reads and displays the register image save area that was constructed on entry into the monitor. (This is 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 by writing the "exit monitor" command (8001H) to **MONITOR_CONTROL**. This 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 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 monitor, the host issues the error message "cannot break into monitor." The "are_you_there?" function is used to see whether the 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. (This is 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 monitor by 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.

A

Emulation Error Messages

68030 Emulation Error Messages

This appendix lists the 68030 emulator 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. The messages are in alphabetical order.

Attempt to read guarded memory, addr = XXXX

The processor tried to read a memory location mapped as "guarded." The offending address is displayed in the XXXX field.

Attempt to write guarded memory, addr = XXXX

The processor tried to modify a memory location mapped as "guarded." The offending address is displayed in the XXXX field.

cannot break into monitor

The host expects to find the CPU executing the monitor, but the "are_you_there?" function shows 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 find the problem, define an analysis trace to trigger on the acknowledge cycle for the level 7 interrupt:

```
trace trigger_on a= 0fffffffh 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. Then the CPU will not respond to the level 7 interrupt immediately.

The trace list should show an emulator generated jam cycle. MONITOR_ENTRY should be the address supplied by these cycles. Compare the trace list of the monitor entry point to a monitor listing. Ensure 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, the "are_you_there?" function may be timing out.

Next, define a 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 memory system's width.

Ensure 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

The host tried to clear a breakpoint in target system memory, but the emulator could not break to the monitor to clear the breakpoint.

Could not enable breakpoint at address XXXX

The host tried to set a breakpoint in target system memory, but the emulator could not break into the monitor 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 also will cause this error message.

monitor did not respond to exit request

The host expected to find the CPU executing somewhere other than in the monitor, but the "are_you_there" monitor function shows otherwise. This message occurs after issuing a command that causes a return to the user program from the monitor. (For example: display registers while the user program is executing, or "run" while in the monitor, and so on).

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 find the problem, define a 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 (in an interrupt routine or elsewhere) at the time of an "exit monitor" command, the command cannot be recognized. This error message is displayed after a timeout.

Observe the exit mechanism from the monitor, and compare the trace to the monitor listing. Be certain that the monitor has not been inadvertently overwritten.

Once the monitor is exited, make sure 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

You tried 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 software_breakpoints set ...** command. If you used a **modify memory ...** command to set the breakpoint, or if the breakpoint was in the absolute code loaded into the emulator, you can't clear such breakpoints using **modify software_breakpoints clear ...** commands.

(no termination) message in tracelist

A particular CPU cycle was terminated by LBERR or LHALT instead of the usual termination by DSACKs or STERM.

This message also can be a clue that the target system is violating the MC68030 specification that 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

The emulator has not received a low-to-high or high-to-low transition on the 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. For example, if a DMA mechanism 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")

The CPU is being reset by the emulator.

reset (with lower case "r")

The CPU is being reset by target system hardware.

running The emulator is running in a user program.

running in monitor The emulator is running in the monitor.

slow dev at a= XXXX (YY) The CPU is trying 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 is the address of the attempted cycle, and the YY field is the function code applied to the cycle according to the following table:

YY Field Value	Meaning
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.

SRU Error Messages

See the chapter titled "Using SRU" in the *HP 64000-UX System User's Guide* for information on error and warning messages generated during the SRU build process.

B

Timing Comparisons

Introduction

This appendix contains tables that 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[15]		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.						

MC68030/HP 64430 Timing Comparisons						
13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES						
(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; T_A = 0 to 70 C)						
Num	**Characteristic**	**33.33 MHz[15]**		**HP 64430**		**Unit**
		Min	**Max**	**Min**	**Max**	
6	Clock High to FC, Size, RMC, CIOUT Address Valid Clock High to IPEND Valid	0 0	14 14	0 0	14 24	ns ns
6A	Clock High to ECS, OCS Asserted	0	12	0	15	ns
6B	FC, Size, RMC, CIOUT Address Valid to Negating ECS IPEND Valid to Negating ECS	3 3	--- ---	3 -1	--- ---	ns ns
7	Clock High to FC, Size, RMC, CIOUT, Address, Data High Impedance	0	30	0	30	ns
8	Clock High to FC, Size, RMC, IPEND, CIOUT, Address Invalid	0	---	0	---	ns
9	Clock Low to AS, DS, CBREQ Asserted	2	10	2	15	ns
9A[1]	AS to DS Assertion Skew (Read)	-8	8	-8	8	ns
9B[14]	AS Asserted to DS Asserted (Write)	22	---	22	---	ns
10	ECS Width Asserted	8	---	7	---	ns
10A	OCS Width Asserted	8	---	7	---	ns
10B[7]	ECS, OCS Width Negated	5	---	5	---	ns
11	FC, Size, RMC, CIOUT, Address Valid to AS Asserted (and DS Asserted, Read) IPEND to AS Asserted (and DS Asserted, Read)	5 5	--- ---	5 -1	--- ---	ns ns
12	Clock Low to AS, DS, CBREQ Negated	0	10	0	15	ns
12A	Clock Low to ECS/OCS Negated	0	15	0	16	ns
13	AS, DS Negated to FC, Size, RMC, CIOUT, Address Invalid	5	---	5	---	ns

MC68030/HP 64430 Timing Comparisons						
13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES						
(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; T_A = 0 to 70 C)						
Num	**Characteristic**	**33.33 MHz[15]**		**HP 64430**		**Unit**
		Min	**Max**	**Min**	**Max**	
14	$\overline{AS}$ (and $\overline{DS}$, Read) Width Asserted (Asynchronous Cycle)	45	---	43	---	ns
14A[11]	$\overline{DS}$ Width Asserted, Write	23	---	21	---	ns
14B	$\overline{AS}$ (and $\overline{DS}$ Read) Width Asserted (Synchronous Cycle)	23	---	21	---	ns
15	$\overline{AS}$, $\overline{DS}$ Width Negated	23	---	21	---	ns
15A[8]	$\overline{DS}$ 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[6]	R/$\overline{W}$ High to $\overline{AS}$ Asserted	5	---	5	---	ns
22[6]	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[6.11]	$\overline{AS}$, $\overline{DS}$ Negated to Data Out Invalid	5	---	5	---	ns
25A[9.11]	$\overline{DS}$ Negated to $\overline{DBEN}$ Negated (Write)	5	---	5	---	ns
26[6.11]	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

MC68030/HP 64430 Timing Comparisons

13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES

(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; T_A = 0 to 70 C)

Num	Characteristic	33.33 MHz[15]		HP 64430		Unit
		Min	Max	Min	Max	
28[12]	$\overline{AS}$, $\overline{DS}$ Negated to $\overline{DSACKx}$, $\overline{BERR}$, $\overline{HALT}$, $\overline{AVEC}$ Negated (Asynchronous Hold)	0	30	0	18	ns
28A[12]	Clock Low to $\overline{DSACKx}$, $\overline{BERR}$, $\overline{HALT}$, $\overline{AVEC}$ Negated (Synchronous Hold)	6	50	6	43	ns
29[12]	$\overline{DS}$ Negated to Data-In Invalid (Asynchronous Hold)	0	---	0	---	ns
29A[12]	$\overline{DS}$ Negated to Data-In High Impedance	---	30	---	27	ns
30[12]	Clock Low to Data-In Invalid (Synchronous Hold)	6	---	6	---	ns
30A[12]	Clock Low to Data-In High Impedance (Read followed by Write)	---	45	---	35	ns
31[2]	$\overline{DSACKx}$ Asserted to Data-In Valid (Asynchronous Data Setup)	---	20	---	20	ns
31A[3]	$\overline{DSACKx}$ Asserted to $\overline{DSACKx}$ Valid (Skew)	---	5	---	5	ns
32	$\overline{RESET}$ Input Transition Time	---	1.5	---	1.5	Clks
33	Clock Low to $\overline{BG}$ Asserted	0	15	0	24	ns
34	Clock Low to $\overline{BG}$ Negated	0	15	0	24	ns
35	$\overline{BR}$ Asserted to $\overline{BG}$ Asserted ($\overline{RMC}$ Not Asserted)	1.5	3.5	1.5	3.5	Clks
37	$\overline{BGACK}$ Asserted to $\overline{BG}$ Negated	1.5	3.5	1.5	3.5	Clks
37A	$\overline{BGACK}$ Asserted to $\overline{BR}$ Negated	0	1.5	0	1.5	Clks
39[6]	$\overline{BG}$ Width Negated	45	---	43	---	ns
39A	$\overline{BG}$ Width Asserted	45	---	43	---	ns
40	Clock High to $\overline{DBEN}$ Asserted (Read)	0	18	0	19	ns
41	Clock Low to $\overline{DBEN}$ Negated (Read)	0	18	0	19	ns

MC68030/HP 64430 Timing Comparisons

13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES

(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; T_A = 0 to 70 C)

Num	Characteristic	33.33 MHz[15]		HP 64430		Unit
		Min	Max	Min	Max	
42	Clock Low to $\overline{\mathrm{DBEN}}$ Asserted (Write)	0	18	0	19	ns
43	Clock High to $\overline{\mathrm{DBEN}}$ Negated (Write)	0	18	0	19	ns
44	R/$\overline{\mathrm{W}}$ Low to $\overline{\mathrm{DBEN}}$ Asserted (Write)	5	---	5	---	ns
45[5]	$\overline{\mathrm{DBEN}}$ Width Asserted (Asynchronous Read) DBEN Width Asserted (Asynchronous Write)	30 60	--- ---	28 58	--- ---	ns ns
45[9]	$\overline{\mathrm{DBEN}}$ Width Asserted (Synchronous Read) DBEN Width Asserted (Synchronous Write)	5 30	--- ---	5 28	--- ---	ns ns
46	R/$\overline{\mathrm{W}}$ Width Asserted (Asynchronous Write or Read)	75	---	75	---	ns
46A	R/$\overline{\mathrm{W}}$ Width Asserted (Synchronous Write or Read)	45	---	45	---	ns
47A	Asynchronous Input Setup Time ($\overline{\mathrm{HALT}}$, $\overline{\mathrm{BERR}}$, DSACKx) Asynchronous Input Setup Time ($\overline{\mathrm{IPLx}}$)	2 2	--- ---	9 12	--- ---	ns ns
47B	Asynchronous Input Hold Time from Clock Low	6	---	6	---	ns
48[4]	$\overline{\mathrm{DSACKx}}$ Asserted to $\overline{\mathrm{BERR}}$, $\overline{\mathrm{HALT}}$ Asserted	---	18	---	16	ns
53	Data Out Hold from Clock High	2	---	2	---	ns
55	R/$\overline{\mathrm{W}}$ Asserted to Data Bus Impedance Change	15	---	12	---	ns
56	$\overline{\mathrm{RESET}}$ Pulse Width (Reset Instruction)	512	---	512	---	Clks
57	$\overline{\mathrm{BERR}}$ Negated to $\overline{\mathrm{HALT}}$ Negated (Rerun)	0	---	2	---	ns
58[10]	$\overline{\mathrm{BGACK}}$ Negated to Bus Driven	1	---	1	---	Clks
59[10]	$\overline{\mathrm{BG}}$ Negated to Bus Driven	1	---	1	---	Clks
60[13]	Synchronous Input Valid to Clock High (Setup Time)	2	---	4	---	ns
61[13]	Clock High to Synchronous Input Invalid (Hold Time)	6	---	6	---	ns

MC68030/HP 64430 Timing Comparisons						
13.6 AC ELECTRICAL SPECIFICATIONS -- READ AND WRITE CYCLES						
(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; T_A = 0 to 70 C)						
Num	**Characteristic**	**33.33 MHz[15]**		**HP 64430**		**Unit**
		Min	**Max**	**Min**	**Max**	
62	Clock Low to $\overline{\text{STATUS}}$, $\overline{\text{REFILL}}$ Asserted	0	15	0	25	ns
63	Clock Low to $\overline{\text{STATUS}}$, $\overline{\text{REFILL}}$ 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 $\overline{\text{DSACKx}}$ 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 DSACKx, BERR is an asynchronous input using the asynchronous input setup time (#47A).
5. 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, BG may be reasserted.
7. This specification indicates the minimum high time for ECS and 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 DS negated to 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 DBEN. The timing on 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. 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 CS signal of an MC68881/MC68882 with AS (allowing 7ns for a gate delay) and still meet the CS to 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 $t_1/2$.

HP 64430 DC Electrical Specifications

(Vcc = 5.0 Vdc +/- 5%; GND = 0 Vdc; T_A = 0 to 70 C)

Characteristic		Symbol	Min	Max	Unit
Input High Voltage		V_{IH}	2.0	V_{CC}	V
Input Low Voltage		V_{IL}	-0.5	0.8	V
Input Leakage Current GND = or < Vin = or < VCC	BR, BGACK, IPLx, MMUDIS, CDIS,	I_{in}	-2.5	2.5	uA
Input High Current	CBACK, CIIN, STERM BERR, AVEC, DSACKx, HALT CLK, RESET	I_{IH}	--- --- ---	0 25 50	uA
Input Low Current	RESET, CBACK, CIIN, STERM CLK, BERR, AVEC, DSACKx, HALT	I_{IL}	--- ---	-1.4 -0.25	mA
Output High Voltage I_{OH} = -400uA	A0-A31, AS, BG, D0-D31, DBEN, DS, ECS, R/W, STATUS, REFILL, IPEND, OCS, RMC, SIZ0-SIZ1, FC0-FC2, CBREQ, CIOUT	V_{OH}	2.4	---	V
Output Low Voltage I_{OL} = 2.5mA I_{OL} = 3.2 mA I_{OL} = 4.5mA I_{OL} = 5.3 mA I_{OL} = 2.0 mA I_{OL} = 9.3 mA	 A0-A31, FC0-FC2, SIZ0-SIZ1 BG, D0-D31 CBREQ, R/W, RMC AS, DS, DBEN, IPEND STATUS, REFILL, CIOUT, ECS, OCS RESET	V_{OL}	 --- --- --- --- --- ---	 0.5 0.5 0.5 0.5 0.5 0.5	V
Power Dissipation	T_A = 0 C T_A = 70 C	P_D	--- ---	0 0	W
Capacitance (V_{IN} = 0V, T_A = 25 C, f = 1MHz)		C_{in}	---	20	pF
Load Capacitance		C_L	--- ---	100 50	pF

Index

A

absolute files, loading, **6-24**
accessing the emulation system, **3-9**
address range overlays, **4-16**
address specification, custom register, **8-3**
address translation cache (ATC), flushing, **5-3**
analysis with cache enabled, **6-13**
analyzer, **6-26 - 6-27**
analyzer, debugging plug-in failures, **6-26**
analyzer, use for debugging, **6-26**
are you there function, how it works, **10-1**
asterisk in memory display, **3-33**

B

background, **1-5**
background monitor, **7-2**
background monitor, defaulting stack pointer for, **4-8**
BERR, enabling/disabling, **6-16**
BKPT 7 shown in memory display, **3-33**
block size, **4-15**
blocking target BERR during emulation memory cycles, **4-12**
break function, breaking into the monitor, **7-5**
break on write to ROM, **1-3**
breakpoints, **1-4**
bus cycle, **6-26 - 6-27**
bus cycle, none for more than 250 milliseconds, **6-27**
bus size bit (B), **6-26**

C

cache control, **6-12**
caches, using the, **6-12**
card cage access cover, removing the, **2-6**
code branches, incorrect, **6-26**
command modules, emulation monitor, **7-8**
command scanner, **7-8**
comparison of foreground/background monitors, **7-2**
configuration file name, **4-32**
configuration file, .EA, **2-12**
configuration file, .EB, **2-12**
configuration file, starting with the default, **6-26**

configuration file, viewing, **6-26**
configuration file, what to do after modifying, **2-12**
configuration file, incorrect, **6-26**
configuration, deMMUer, **4-27**
configuration, review, **6-25**
configuring simulated I/O, **9-1**
connecting the emulator pod to your target system, **2-9**
continuing target system interrupts while in the emulation monitor, **7-15**
controlling flow of data and code, **4-13**
coprocessor configuration questions, **8-10**
coprocessor copy routine, **8-9**
coprocessor register buffer, emulation monitor, **8-7**
copying from target memory, how it works, **10-15**
copying the demonstration programs to your subdirectory, **3-8**
copying to target system memory, how it works, **10-16**
cpu registers, modifying, how it works, **10-17**
custom coprocessor, register set display specification, **8-4**
custom coprocessors support, **1-4**
custom coprocessors, modifying configuration for, **4-10**
custom register address specification, **8-3**
custom register format file, **8-2**
custom register format file, specifying, **4-10**
custom register name specification, **8-4**
custom register size specification, **8-3**
custom registers, emulation monitor changes, **8-7**
customizing the emulation monitor, **7-10**

D

debugging plug-in failures with analyzer, **6-26**
debugging plug-in problems, **6-25**
debugging, use status messages, **6-27**
default response to emulation configuration questions, **4-2**
defaulting stack pointer for background monitor, **4-8**
deleting memory map entries, **4-27**
deMMUer:
- addresses for which it has no translation, **5-3**
- how it operates, **5-2**
- how to access the deMMUer configuration display, **5-8**
- how to turn off, **5-5**
- how to turn on, **5-5**
- startup with the emulator, **5-5**
- turn on/off by setting the analysis mode, **5-6**

turn on/off using emulation configuration questions, **5-6**
what it is, **5-2**
when to start, **5-5**
when to turn off, **5-4**
when to use, **5-3**
deMMUer configuration, **4-27**
dequeued trace list, how to obtain, **3-31**
dequeued trace lists, **3-31**
display memory with source_file symbols, **3-20**
display of trace list dequeued, how to obtain, **3-31**
display source-file command, how to use, **3-12**
displaying cpu registers, how it works, **10-16**
displaying global symbols, **3-14**
displaying local symbols, example, **3-15**
displaying memory, **3-19**
displaying registers, **3-24**
displaying source lines in memory displays, **3-21**
displaying target memory, how it works, **10-13**
dividing the processor address space, **4-13**
DMA enable/disable, **6-16**
DMA transfers into emulation memory, **4-30**
DMA transfers, enabling, **4-29**
DSACK and STERM, interlocking emulation memory and target, **6-8**
DSACK signal problems, open collector drivers on DSACK line, **6-10**
DSACK signal problems, early removal of DSACK signals, **6-10**
DSACK signal problems, isolating the problem, **6-11**
DSACK signal problems, target system, **6-10**
DSACK signals, using, **6-8**

E

.EA configuration file, **2-12**
.EA file, **6-26**
.EB configuration files, **2-12**
emulation configuration, modifying the default, **3-10**
emulation features:
custom coprocessors support, **1-4**
foreground or background monitor, **1-5**
function codes, **1-4**
memory management, **1-4**
out-of-circuit or in-circuit emulation, **1-5**
emulation memory, **4-16**

emulation memory breakpoints with cache enabled, **6-14**
emulation memory display operations, **4-16**
emulation memory load operations, **4-16**
emulation memory, loading, **3-11**
emulation monitor:
  foreground or background, **1-5**
  monitor, **1-5**
emulation monitor changes for custom coprocessors, **8-7**
emulation monitor description, **7-5**
emulation monitor entry point routines, **7-6**
emulation monitor functions, enabling, **4-4**
emulation monitor memory requirements, **4-16**
emulation monitor, coprocessor register buffer, **8-7**
emulation monitor, loading, **6-21, 7-19**
emulation pod configuration, modifying, **4-28**
emulation system components, example system, **3-1**
emulation system hardware, installing, **2-5**
emulation system, accessing, **3-9**
emulation system, preparing, **3-8**
emulator, its purpose, **1-1**
emulator features:
  breakpoints, **1-4**
  processor reset control, **1-4**
  register display/modify, **1-3**
  restrict to real-time runs, **1-2**
  single-step processor, **1-3**
emulator pod cables, connecting to the emulator boards, **2-7**
emulator pod, connecting to the target system, **2-9**
emulator use of int7, enabling, **4-6**
emulator use of software breakpoints, enabling, **4-7**
enabling the foreground monitor, **4-9**
ending the emulation session, **3-37**
ending the mapping session, **4-28**
entering mapper blocks, **4-16**
entering mapper blocks, syntax, **4-17**
entry point routines, emulation monitor, **7-6**
error messages, **A-1**
examples, emulation system used for, **3-1**
exception vector table, **7-6**
EXCEPTION_ENTRY emulation monitor routine, **7-7**

executing a software breakpoint, how it works, **10-6**
external hardware features of the instrumentation card cage, **2-1**

F failures, plug-in, **6-26**
flush the address translation cache, **5-3**
foreground, **1-5**
foreground monitor, **7-3**
foreground monitor, enabling, **4-9**
foreground monitor, interlock or provide termination for, **4-9**
function codes, **1-4**

G getting started procedure, **3-1**
guarded memory access, **4-3**

H halt, **6-26**
hardware installation instructions, **2-5**
highlighted Next PC instruction in memory, **3-27**
how does a simulated interrupt function, **9-5**
HP64000 file format, **3-15**

I I/O operations, **4-16**
IEEE-695 file format, **3-17**
illegal conditions, **4-3**
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, **2-12**
installing software updates, **2-12**
instructions on installing hardware, **2-5**
interlock the foreground monitor, **4-9**
interlocking emulation memory DSACK and STERM and target DSACK and STERM, **6-8**
ipend, enabling target line during emulator breaks, **4-6**
isolate problems, **6-25**

J JSR_ENTRY emulation monitor routine, **7-7**

L leading zeros, **4-19**
linker listing file, example, **7-18**
linking the emulation foreground monitor, **7-19**
loading emulation memory, **3-11**
loading the emulation monitor, **6-21, 7-19**

logical (virtual) address to physical address translation, **5-2**
logical format for trace lists, **3-31**

M making a subdirectory for your 68030 project, **3-2**
mapper blocks, syntax for entering, **4-17**
mapping display softkey labels, **4-14**
mapping memory, **4-13**
memory access timing issues, **6-23**
memory default, **4-15**
memory display, adding source-file symbols, **3-20**
memory displays with source-file code shown, **3-21**
memory management, **1-4**
memory map definition, **4-15**
memory map display entries, **4-14**
memory map example, **4-21**
memory when using the step function, **3-27**
MMU table walks, deMMUer tracking, **5-2**
modify default memory, **4-26**
modify defined_codes, **4-23**
modify memory configuration?, **4-11, 4-29**
modifying a memory configuration, **4-11**
modifying memory, **3-22**
modifying target memory, how it works, **10-15**
modifying the cpu registers, how it works, **10-17**
modifying the default emulation configuration, **3-10**
modifying the emulation monitor exception vector table, **7-11**
modifying the emulation monitor to use simulated interrupts, **9-10**
modifying the memory map, **4-23**
modifying the MON_ALT_BUFFER table, **8-8**
modifying the MON_ALT_REGISTERS table, **8-8**
monitor (emulation):
  background, **7-2**
  comparison of foreground/background, **7-2**
  foreground, **7-3**
monitor message routine, example, **7-16**
MONITOR_ENTRY emulation monitor routine, **7-7**
MONITOR_ENTRY, foreground monitor label, **1-3**
msinit, **2-12**

N name specification, custom register, **8-4**
naming the configuration file, **4-32**
NMI, **7-5**
nosymbols, **3-14**

O on-chip cache disabling, **4-31**
operational overview, **3-1**

P partitioning the processor address space, **4-15**
performance, **6-27**
physical address to logical address translation, **5-2**
plug-in failures, **6-26**
plug-in problems, debugging, **6-25**
pod cable, securing, **2-9**
preinstallation inspection, **2-4**
preparing the emulation system, **3-8**
preparing your program modules, getting started, **3-6**
problems, isolate, **6-25**
purpose of the emulator, **1-1**

R real-time runs, **1-2**
real-time/nonreal-time run mode, selecting, **4-3**
register display/modify, **1-3**
register set display specification, custom coprocessor, **8-4**
removing the development environment card cage access cover, **2-6**
reserved address space, using function codes with, **6-15**
reset control, **1-4**
RESET_ENTRY emulation monitor routine, **7-7**
resetting into the monitor, **4-5, 6-21**
restoring the processor interrupt mask, **7-15**
restrict to real-time runs, **1-2**
restrictions on using simulated I/O, **9-4**
restrictions on using simulated interrupts, **9-9**
run command after a software breakpoint, how it works, **10-7**
run command, how it works, **10-2**
run from ... until command, how it works, **10-4**
run from ... until command, using, **6-19**
run from command, how it works, **10-3**
run until command, how it works, **10-4**
running from the transfer address, **3-23**

S safety considerations, **1-1, 2-3**
securing the pod cable, **2-9**
sending messages from the user program to the emulator display, **7-16**
shutdown, **6-26**
simulated I/O, configuring, **9-1**
simulated I/O, restrictions on using, **9-4**
simulated interrupt, how they function, **9-5**
simulated interrupts, **9-4**
simulated interrupts, modifying the monitor to use, **9-10**
simulated interrupts, restrictions on using, **9-9**
single stepping with background monitor, how it works, **10-10**
single stepping with foreground monitor, how it works, **10-8**
single-step processor, **1-3**
size specification, custom register, **8-3**
software breakpoint instruction number selection, **4-7**
software breakpoint, setting, **10-6**
software breakpoint, shown in memory display, **3-33**
software breakpoints, **1-4, 10-5**
software breakpoints, using, **3-32**
source lines shown in memory displays, **3-21**
source_file display, how to obtain, **3-12**
SRU (Symbolic Retrieval Utilities), **3-13, A-6**
srubuild, **3-13**
stack pointer for background monitor, defaulting, **4-8**
starting address of a block boundary, **4-19**
status conditions, incorrect, **6-26**
status messages, use for debugging, **6-27**
step function, using, **3-26**
stepping through program memory display, **3-27**
STERM signals, using, **6-8**
SWBK_ENTRY emulation monitor routine, **7-7**
symbol handling, **3-13**
symbolic debugging, **1-2**
symbols, **1-2**
symbols, how to add to memory display, **3-20**

T table search, **5-2**
target memory, **4-16**
target memory breakpoints with cache enabled, **6-14**
target memory display operations, **4-16**
target memory load operations, **4-16**

target memory transfers, how they work, **10-11**
target memory, copying from, how it works, **10-15**
target memory, modifying, how it works, **10-15**
target system, **1-1**
target system interface, **6-1**
target system memory, copying to, how it works, **10-16**
target system program interrupt, **7-5**
target system, connecting to the emulator pod, **2-9**
terminate the foreground monitor, **4-9**
timing issues, memory access, **6-23**
trace list dequeuing, example, **3-31**
tracing processor activity, **3-29**
transfer address, running from, **3-23**
translation tables, **5-2**
translation, logical (virtual) address to physical address, **5-2**
translation, physical address to logical address, **5-2**
trigger pulse, **6-27**

U

unpacking the equipment, **2-4**
using breakpoints with caches enabled, **6-13**
using simulated I/O, example, **3-35**
using the emulation monitor, **6-21**
using the emulator, **3-12**
using the modify memory map command, **4-23**

V

vector base register, use of, **6-11**

W

writing coprocessor copy routines, **8-9**

Notes

HEWLETT PACKARD

Hewlett-Packard
Printed in the USA